Clustering

Sahely Bhadra

1. Introduction to Clustering
2. Different type of clustering methods
3. k-means clustering

Unsupervised learning

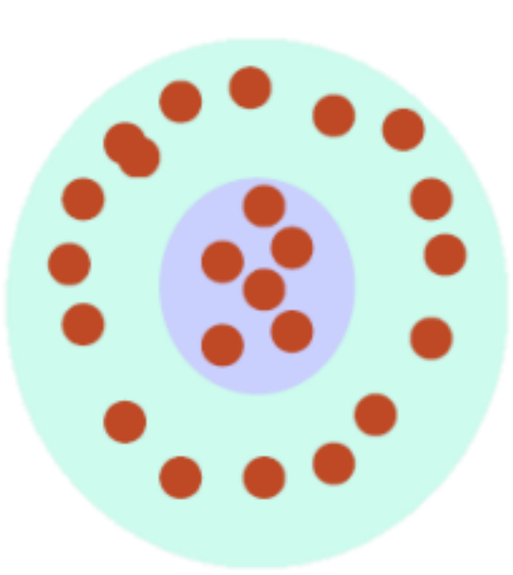
- Machine learning algorithm used to **draw inferences** from datasets consisting of **input data** [x_1,…x_n] **without labeled** responses.
- Goal : The goal is to discover interesting things about the measurements/ data :
    - Is there an informative way to visualise the data?
    - Can we discover subgroups among the features or among the observations/ samples?
- Example :
    - **Cluster analysis** : a broad class of methods for discovering unknown subgroups in data
    - **Principal Component Analysis** : a tool used for data visualization or data pre-processing before supervised techniques are applied : Gene selection, data visualisation

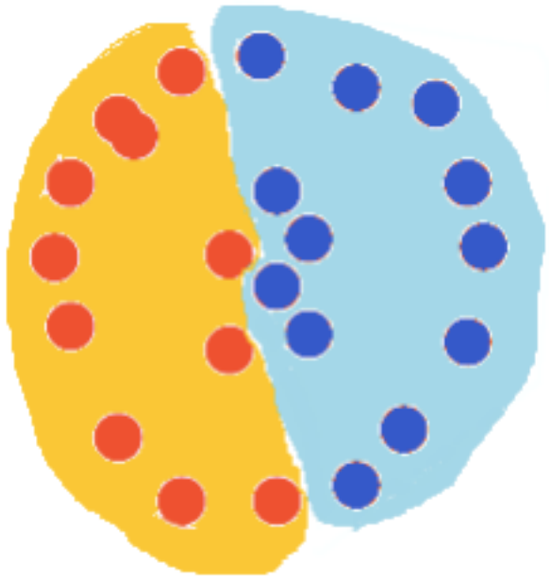
Challenges in unsupervised learning

- Examples :
    - Image segmentation : segment different parts of an image
    - subgroups of breast cancer patients grouped by their gene expression measurements
    - groups of shoppers characterized by their browsing and purchase histories,
    - movies grouped by the ratings assigned by movie viewers.

What is clustering

- The organisation of **unlabelled** data into **similarity** groups called clusters.
- A cluster is a collection of data items which are **“similar” between them**, and **“dissimilar”** to data items in **other clusters**.





Clustering example

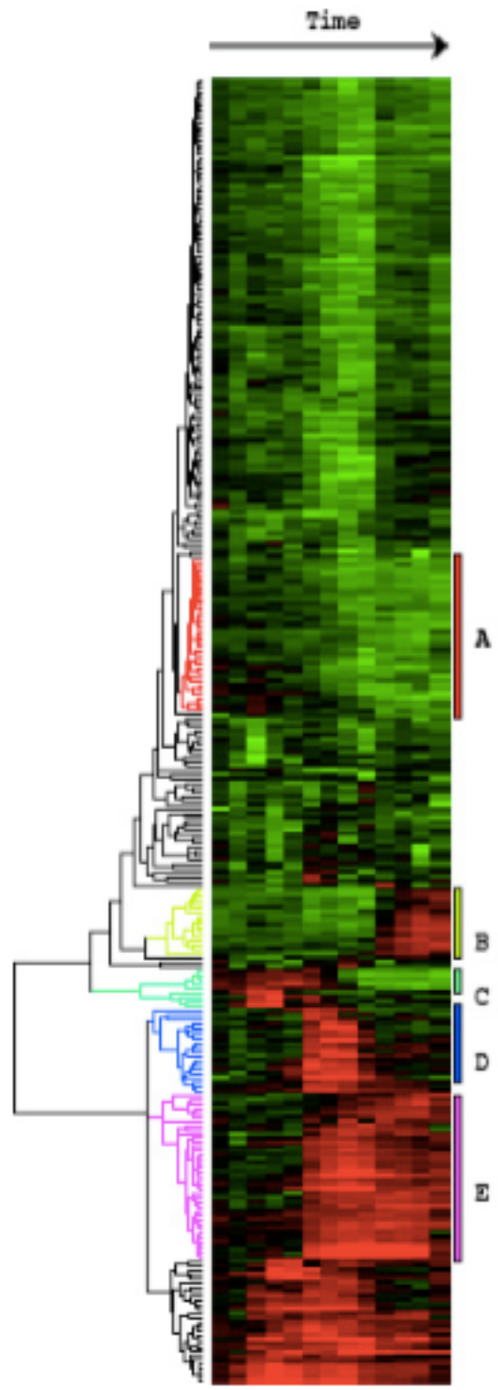
Image segmentation

Goal: Break up the image into meaningful or perceptually similar regions

[Slide from James Hayes]

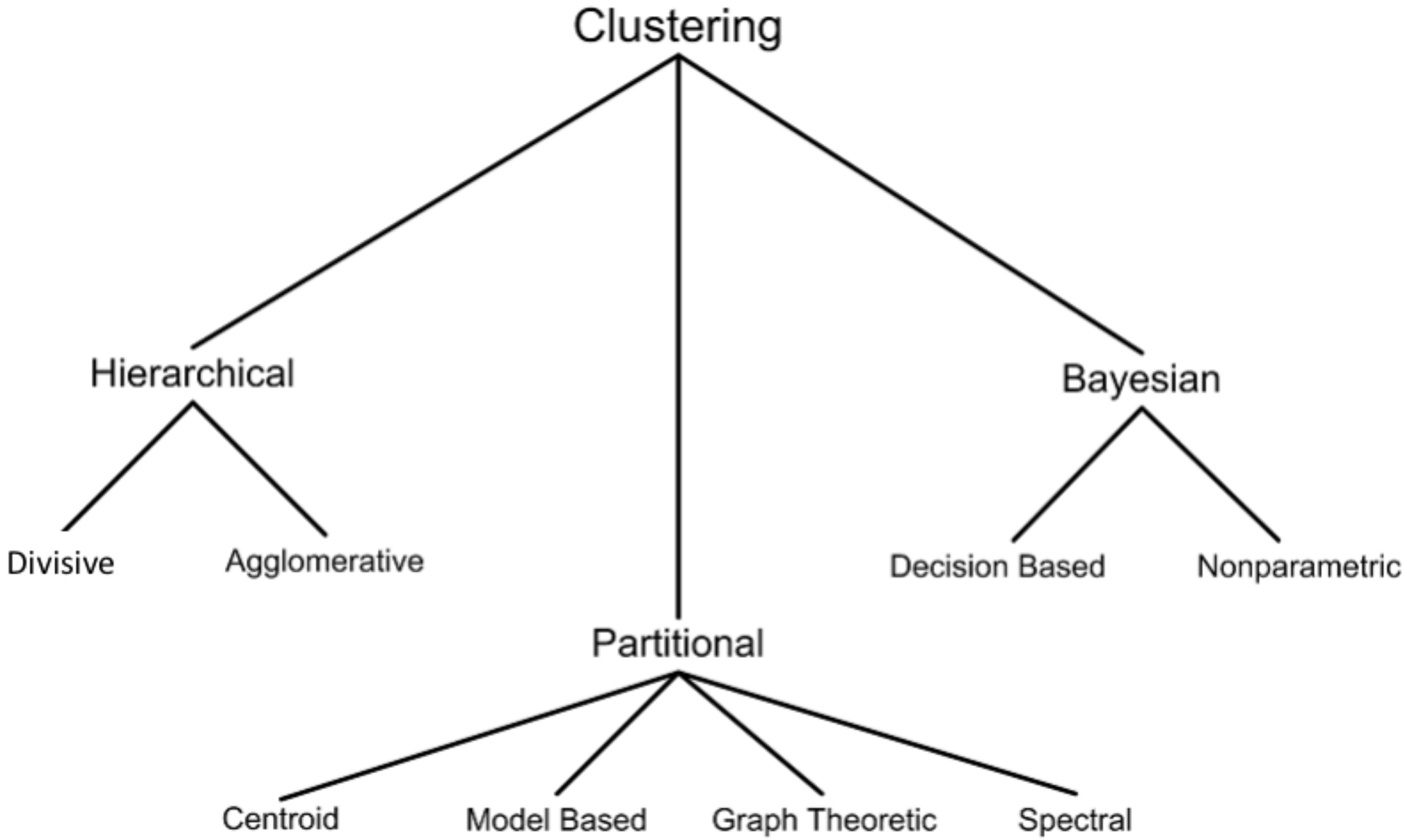
Clustering example

Clustering gene expression data



Eisen et al, PNAS 1998

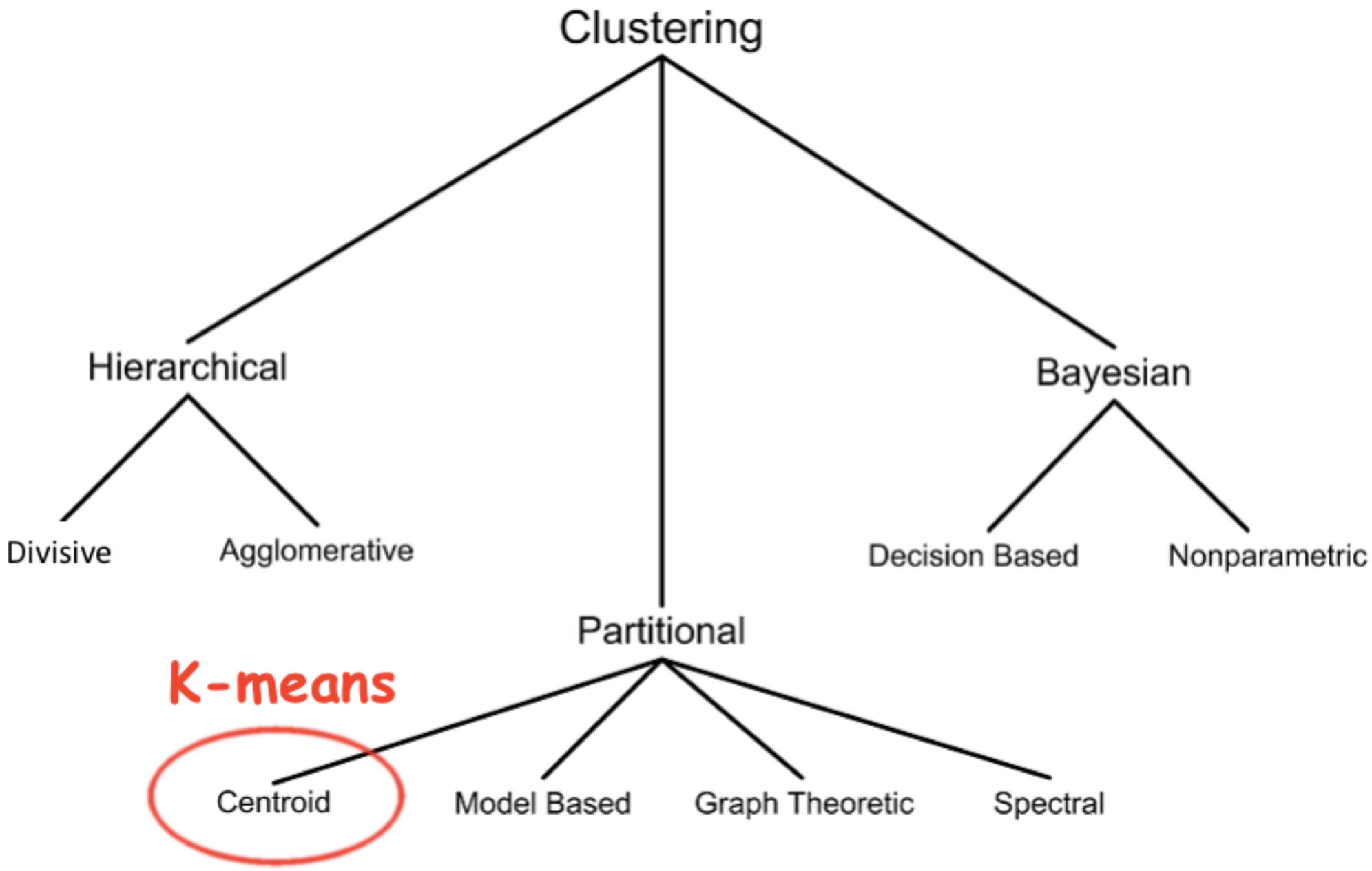
Clustering techniques



Clustering techniques

- **Hierarchical** algorithms find successive clusters using previously established clusters. These algorithms can be either agglomerative ("*bottom-up*") or divisive ("*top-down*"):
  1. Agglomerative algorithms begin with each element as a separate cluster and merge them into successively larger clusters;
  2. Divisive algorithms begin with the whole set and proceed to divide it into successively smaller clusters.
- **Partitional** algorithms typically determine all clusters at once.
- **Bayesian** algorithms try to generate a *posteriori distribution* over the collection of all partitions of the data.

Clustering techniques



K-means Clustering

K-means (MacQueen, 1967) is a partitional clustering algorithm

Let the set of data points D be $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$,

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ is a vector in $X \subseteq R^p$, and p is the number of dimensions.

The *k*-means algorithm partitions the given data into *k* clusters:

- Each cluster has a cluster **center**, called **centroid**.
- *k* is specified by the user

Gaol : Minimising Within cluster variance

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}.$$

K-means Clustering : Algorithm

Given *k*, the *k-means* algorithm works as follows:

1. Choose *k* (random) data points (seeds) to be the initial centroids, cluster centers
2. Assign each data point to the closest centroid
3. Re-compute the centroids using the current cluster memberships
4. If a convergence criterion is not met, repeat steps 2 and 3

K-means Clustering : When to stop

no (or minimum) re-assignments of data points to different clusters, *or*

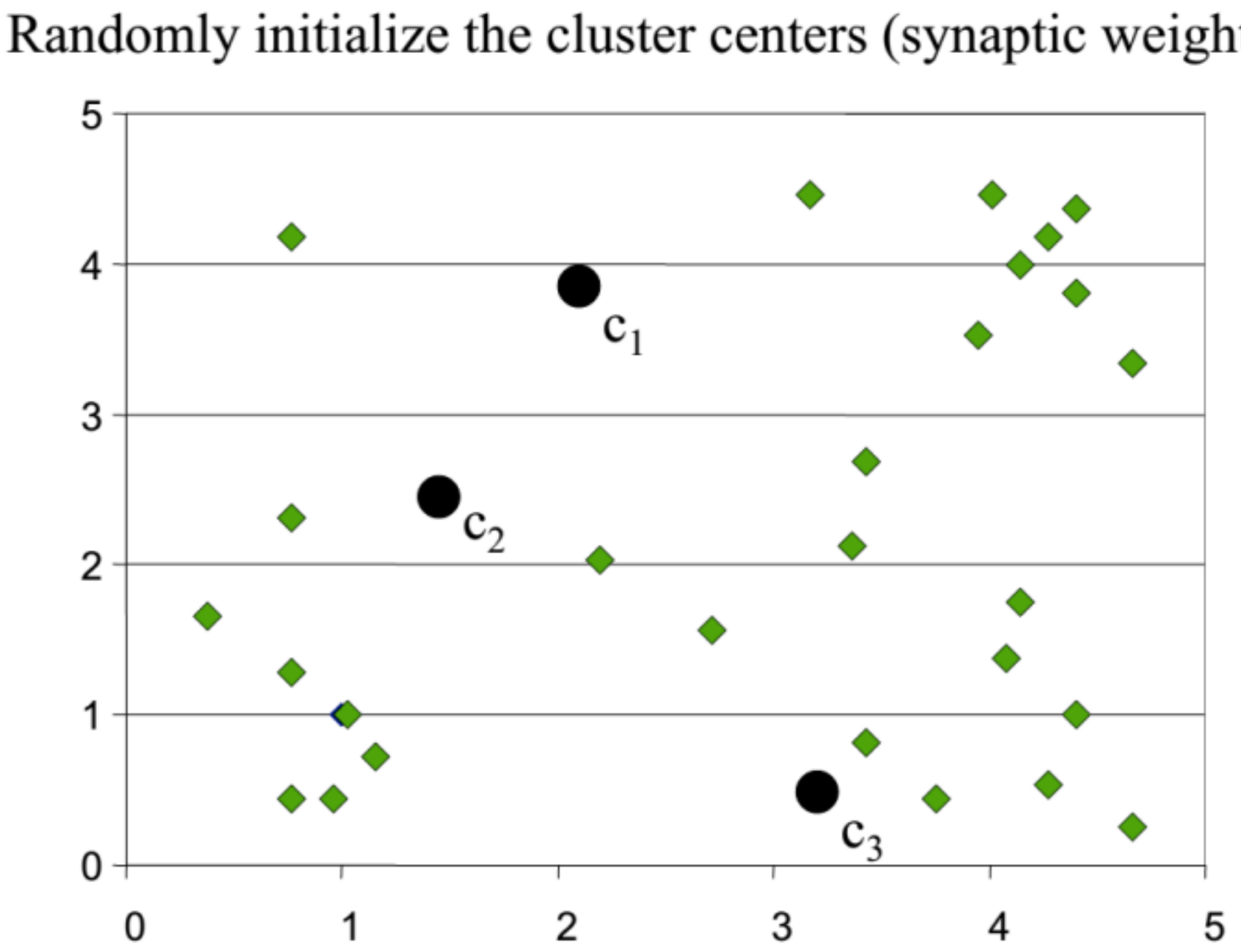
no (or minimum) change of centroids, *or*

minimum decrease in the **sum of squared error** (SSE),

$$SSE = \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

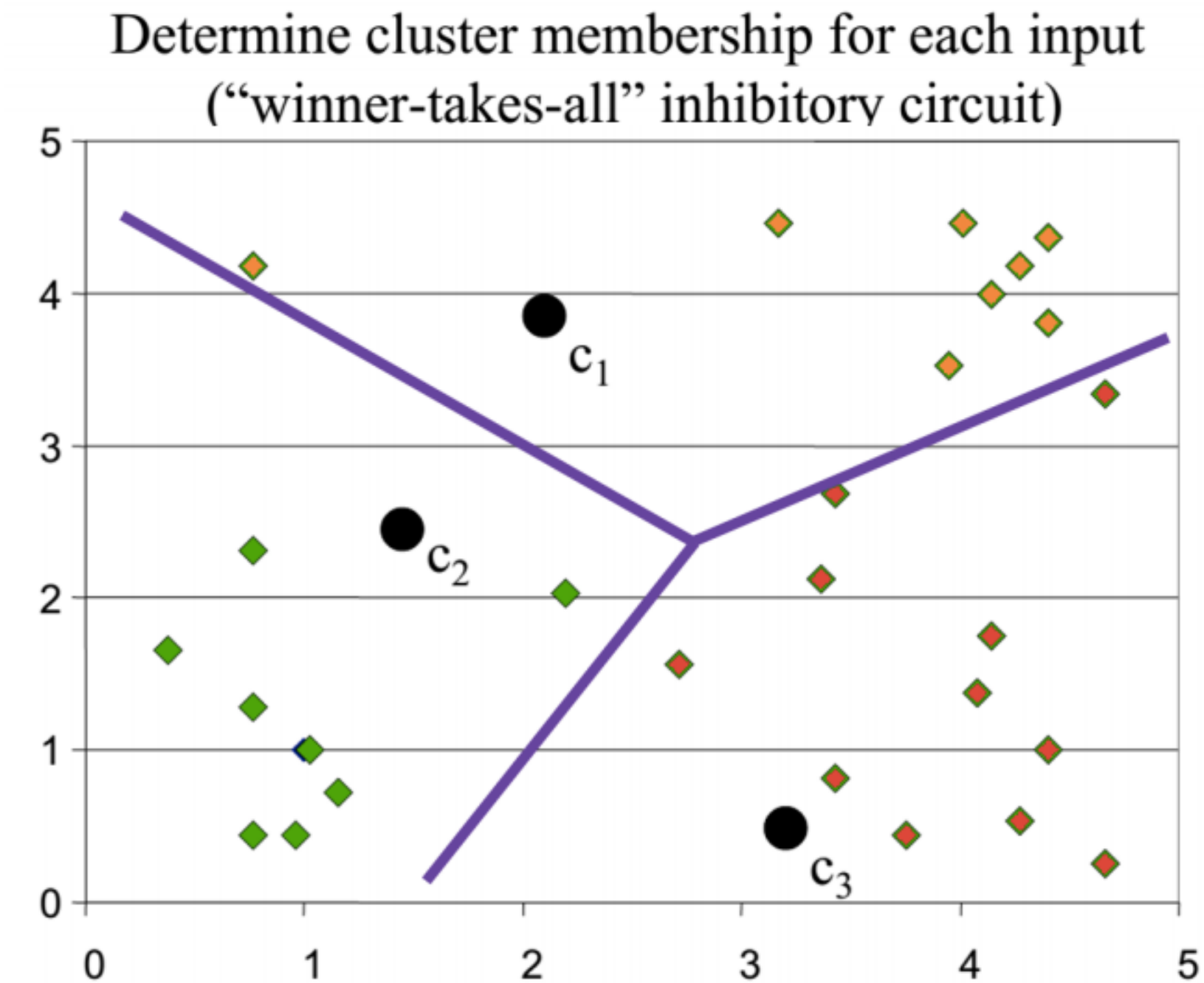
where $\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$ is the mean for feature j in cluster C_k.

K-means Clustering : Example



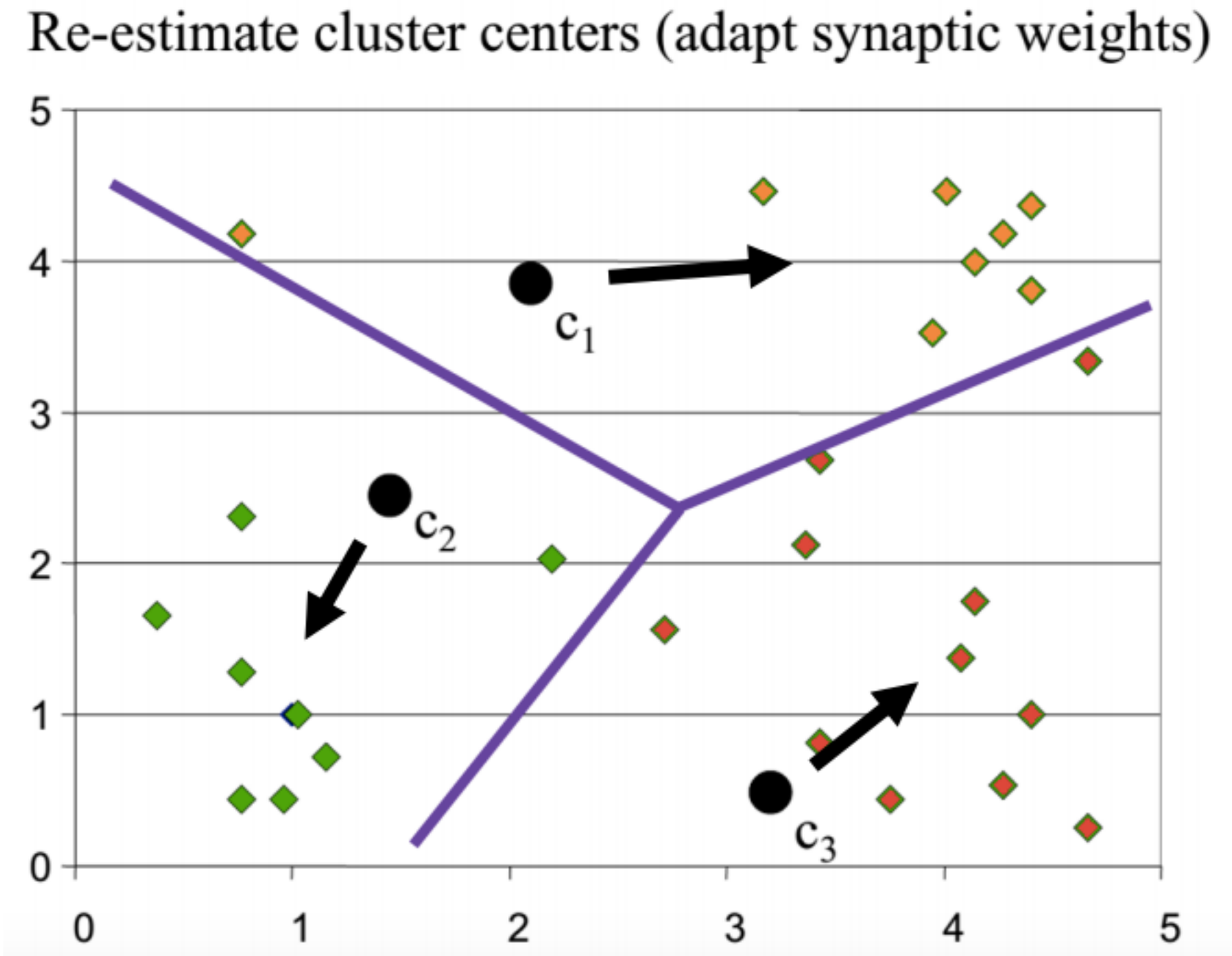
Step1

K-means Clustering : Example



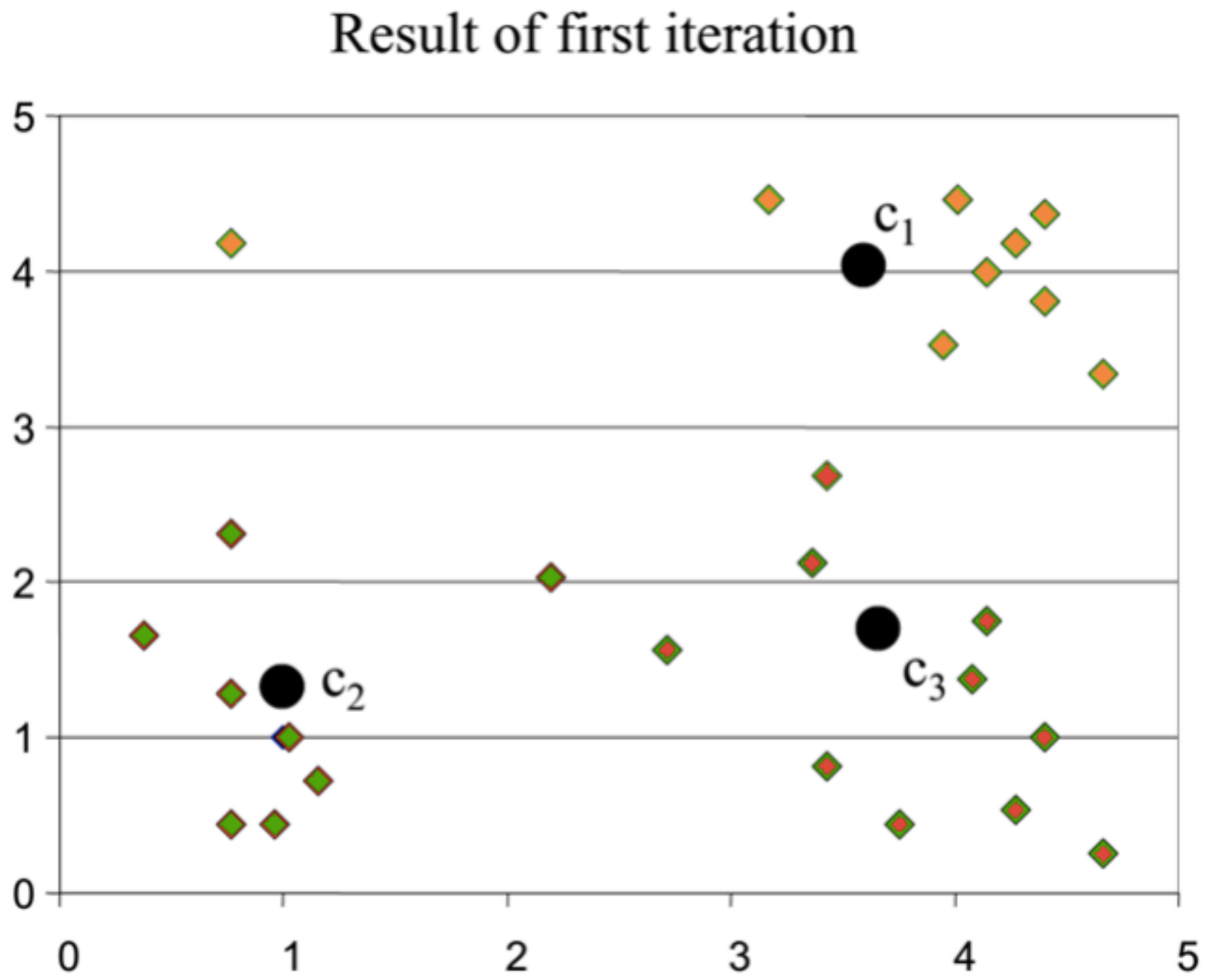
Step2

K-means Clustering : Example

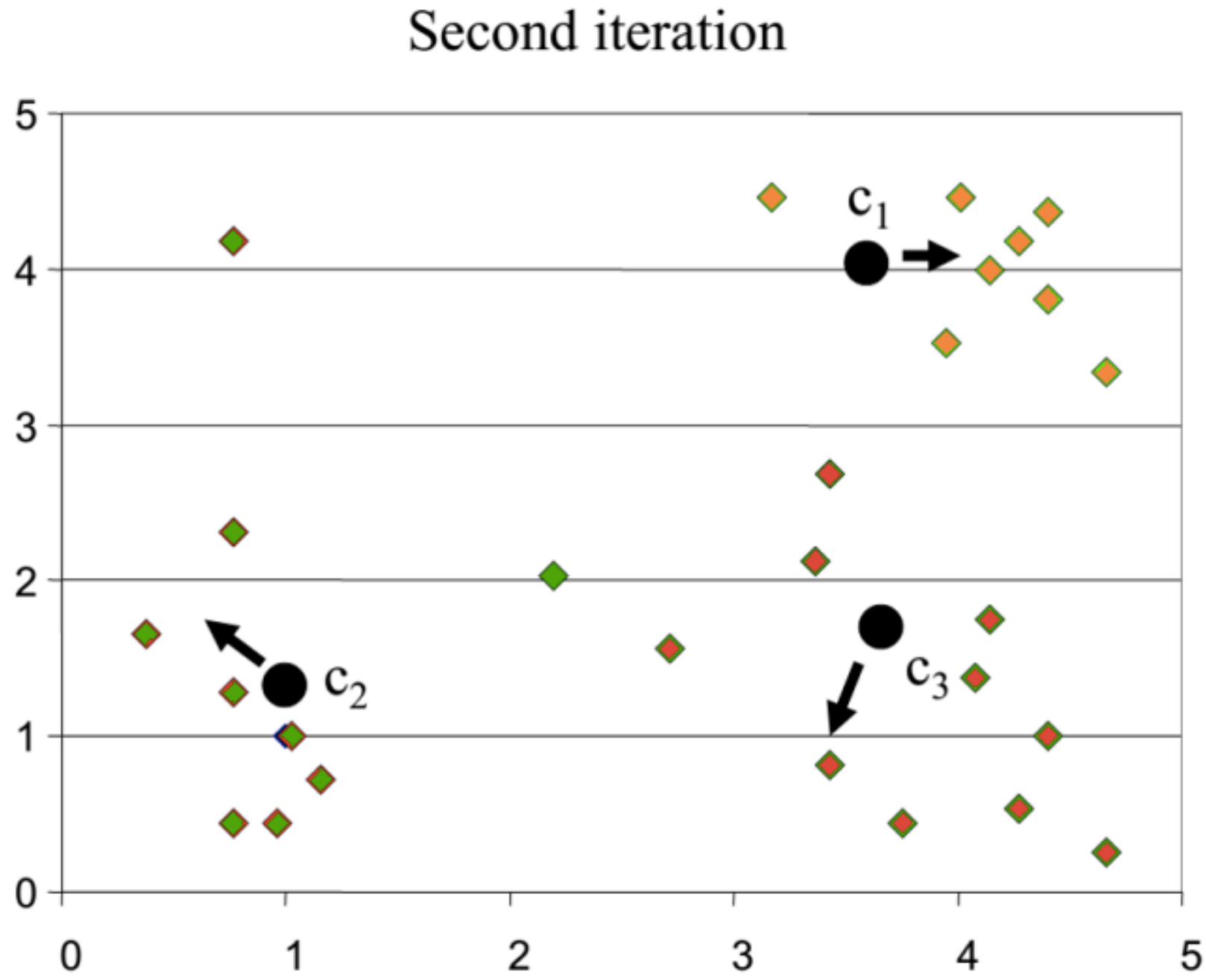


Step3

K-means Clustering : Example

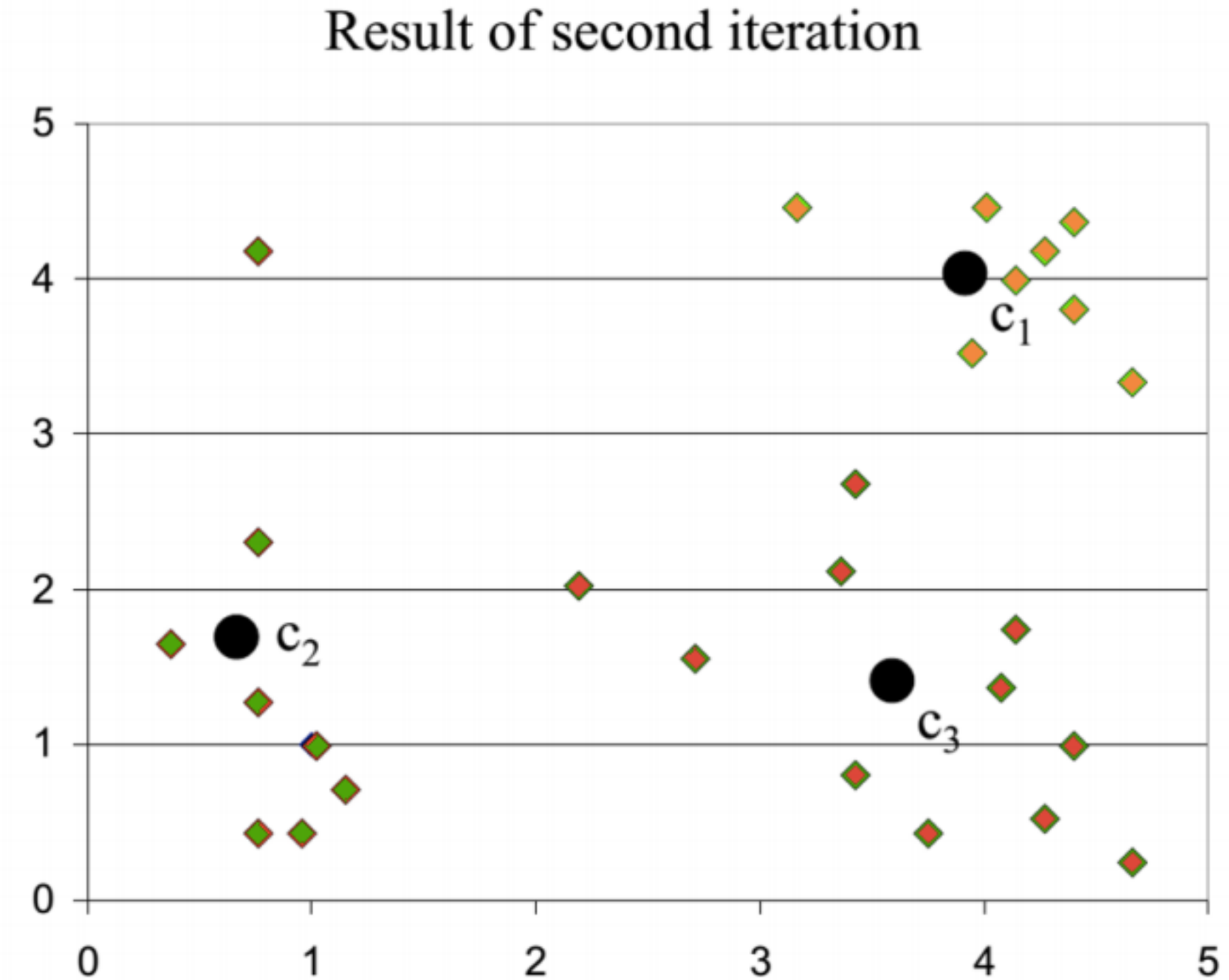


K-means Clustering : Example



Step2

K-means Clustering : Example



Step3

K-means Clustering : Strength

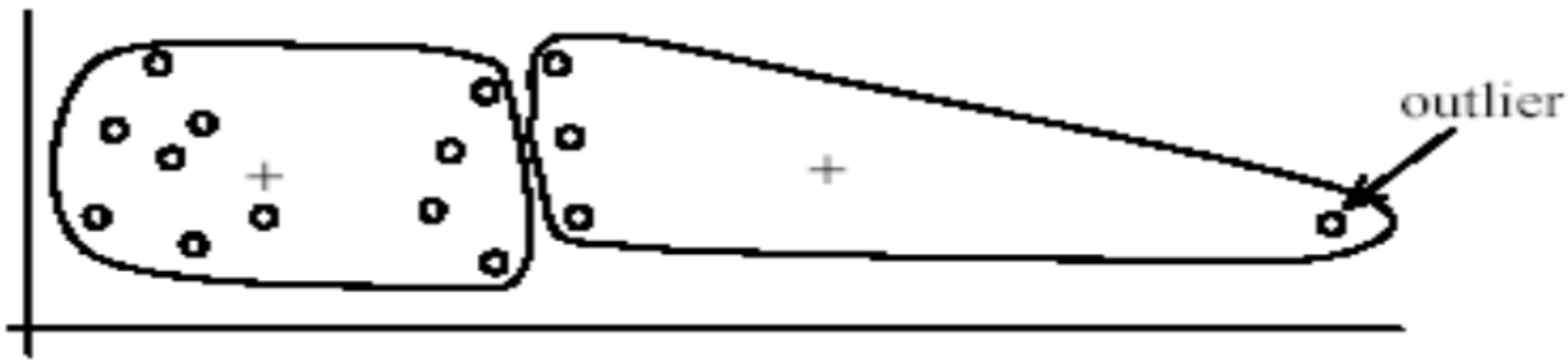
- Strengths:
  - Simple: easy to understand and to implement
  - Efficient: Time complexity: ***O(tkn)***,
    - where n is the number of data points,
    - k is the number of clusters, and
    - t is the number of iterations.
    - Since both k and t are small, k-means is considered an algorithm with linear time complexity.
- K-means is the most popular clustering algorithm.
- Note that: it terminates at a local optimum if SSE is used.
- The global optimum is hard to find due to complexity.

Weakness of K-means Clustering

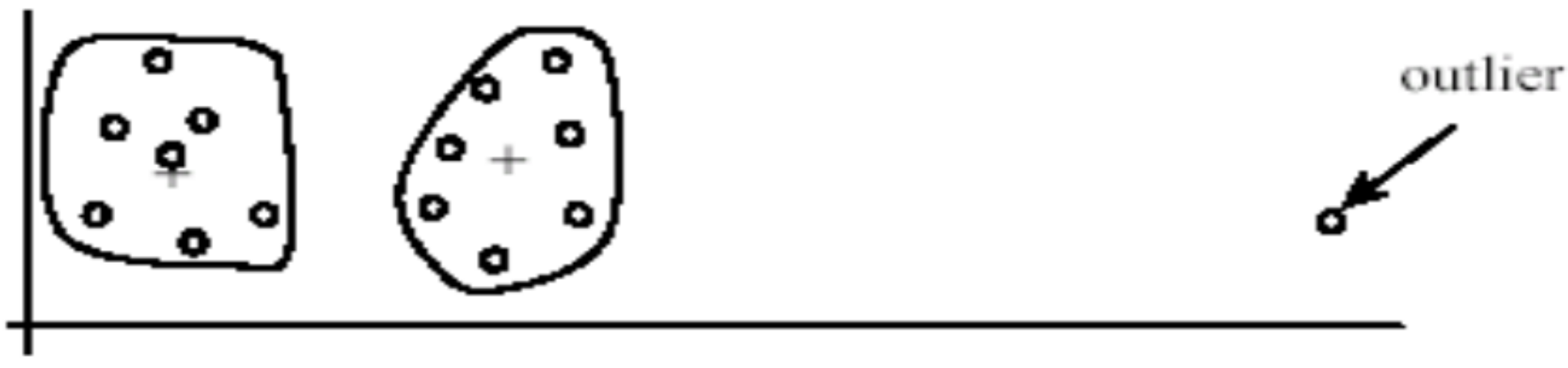
- The algorithm is only applicable if the mean is **defined**.
    - For categorical data, **k-mode** - the centroid is represented by most frequent values.
- The user needs to **specify** k.
- The algorithm is sensitive to **outliers**
        - Outliers are data points that are very far away from other data points.
        - Outliers could be errors in the data recording or some special data points with very different values.
- Output is sensitive to **initial means**

Weakness of K-means Clustering

Sensitive to outliers



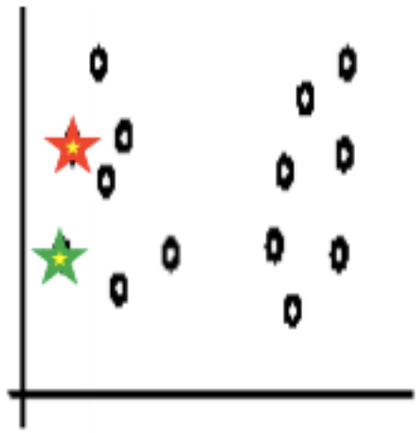
(A): Undesirable clusters



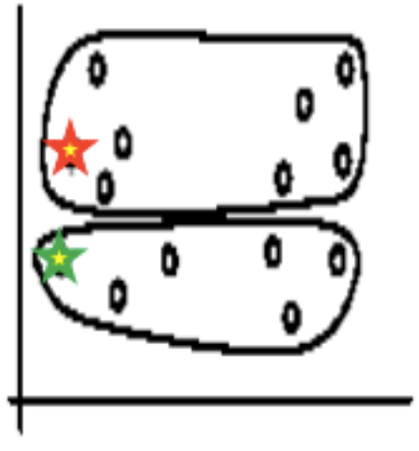
(B): Ideal clusters

Weakness of K-means Clustering

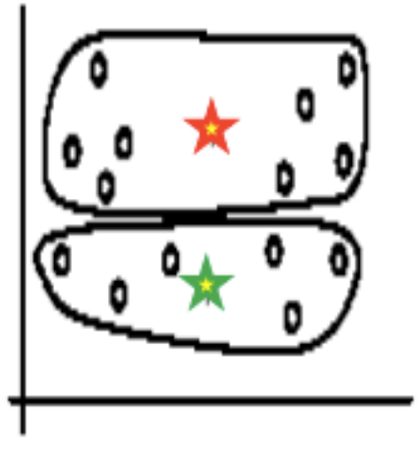
Sensitive to initial points



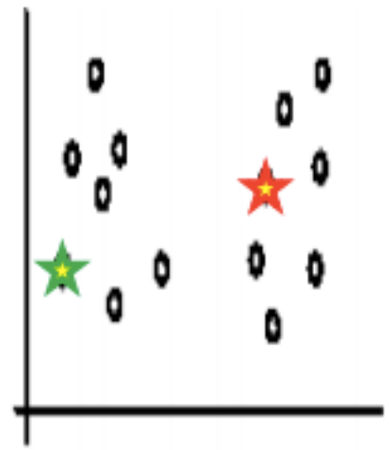
Random selection of seeds (centroids)



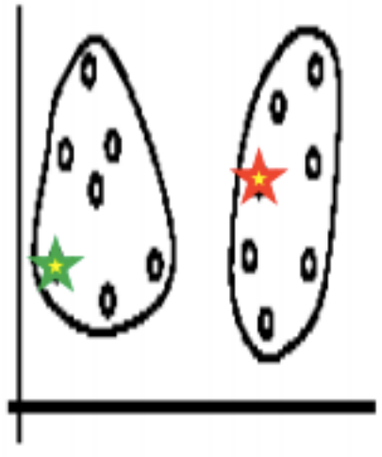
Iteration 1



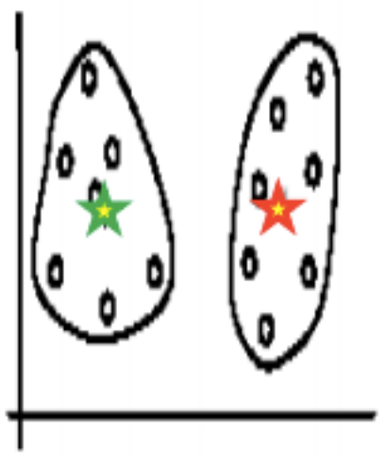
Iteration 2



Random selection of seeds (centroids)



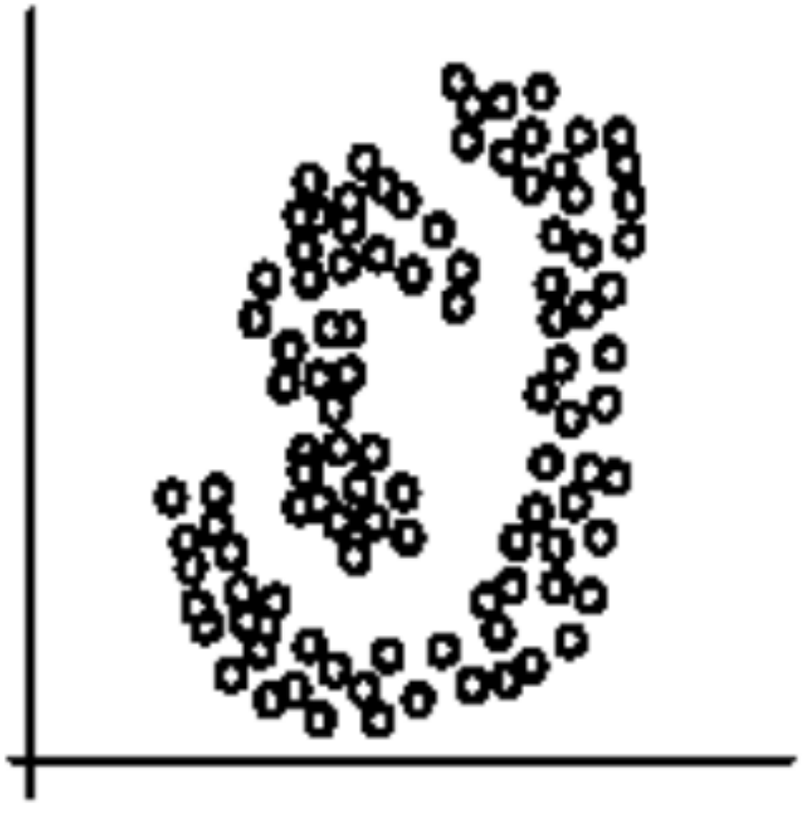
Iteration 1



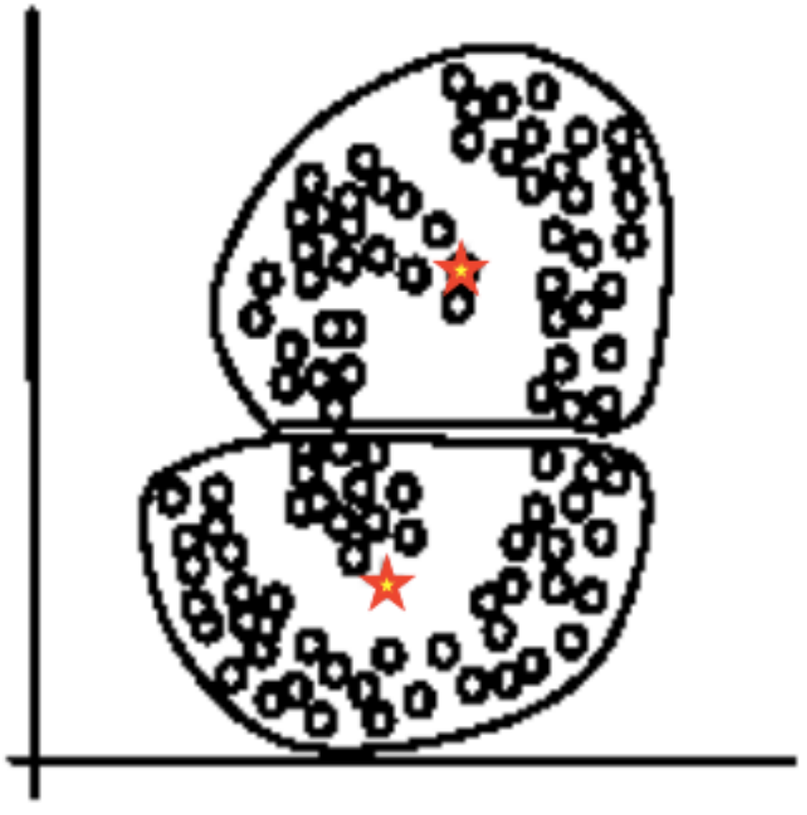
Iteration 2

Weakness of K-means Clustering

Not able to find non-hyper-ellipsoidal cluster



(A): Two natural clusters



(B): *k*-means clusters

Summery K-means Clustering

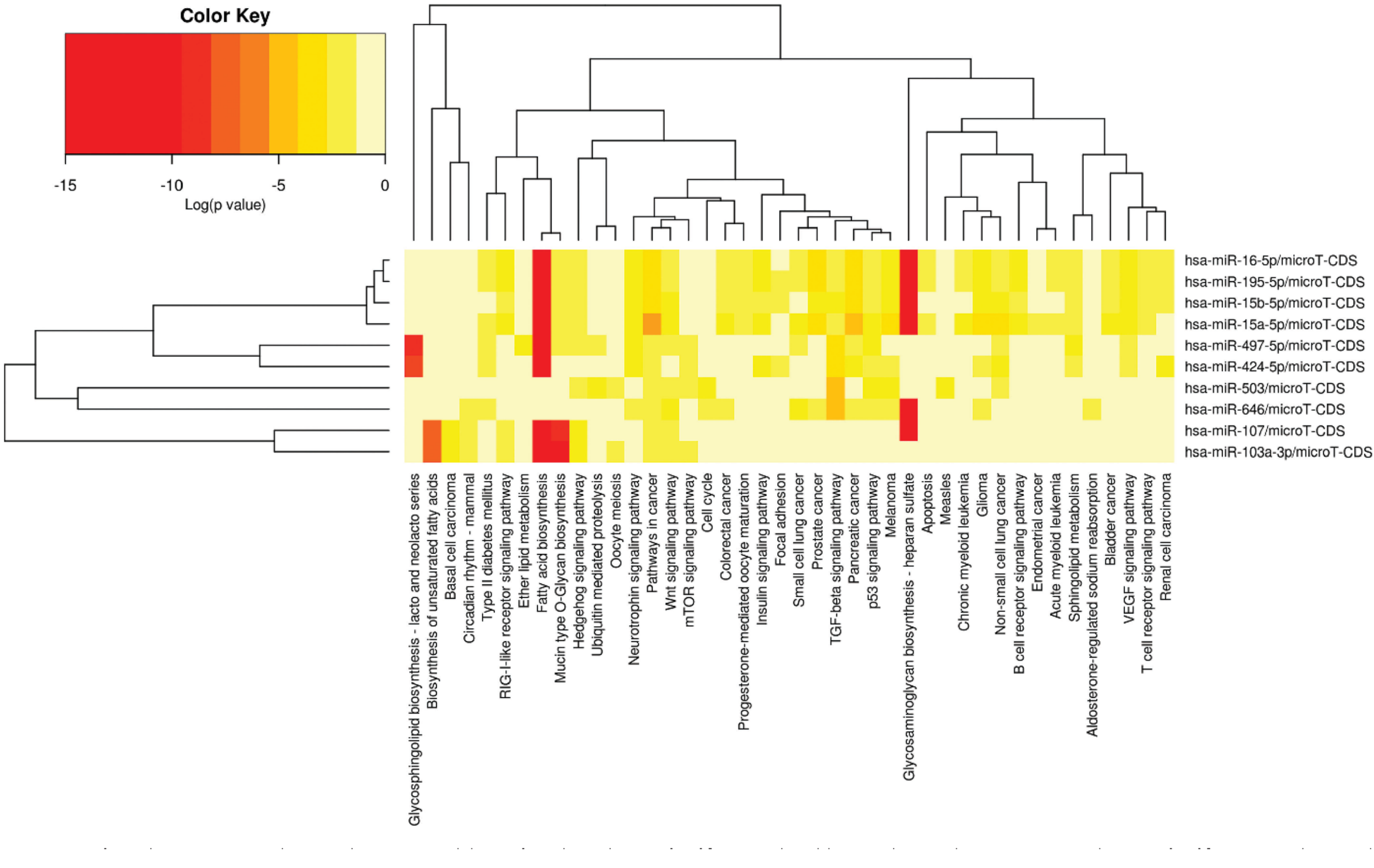
- Despite weaknesses, k-means is still the most popular algorithm due to its simplicity and efficiency
- No clear evidence that any other clustering algorithm performs better in general
- Comparing different clustering algorithms is a difficult task. No one knows the correct clusters!

Hierarchical Clustering

Sahely Bhadra

1. Hierarchical cluster : Agglomerative and Divisive
2.BIRCH

Hierarchical Clustering



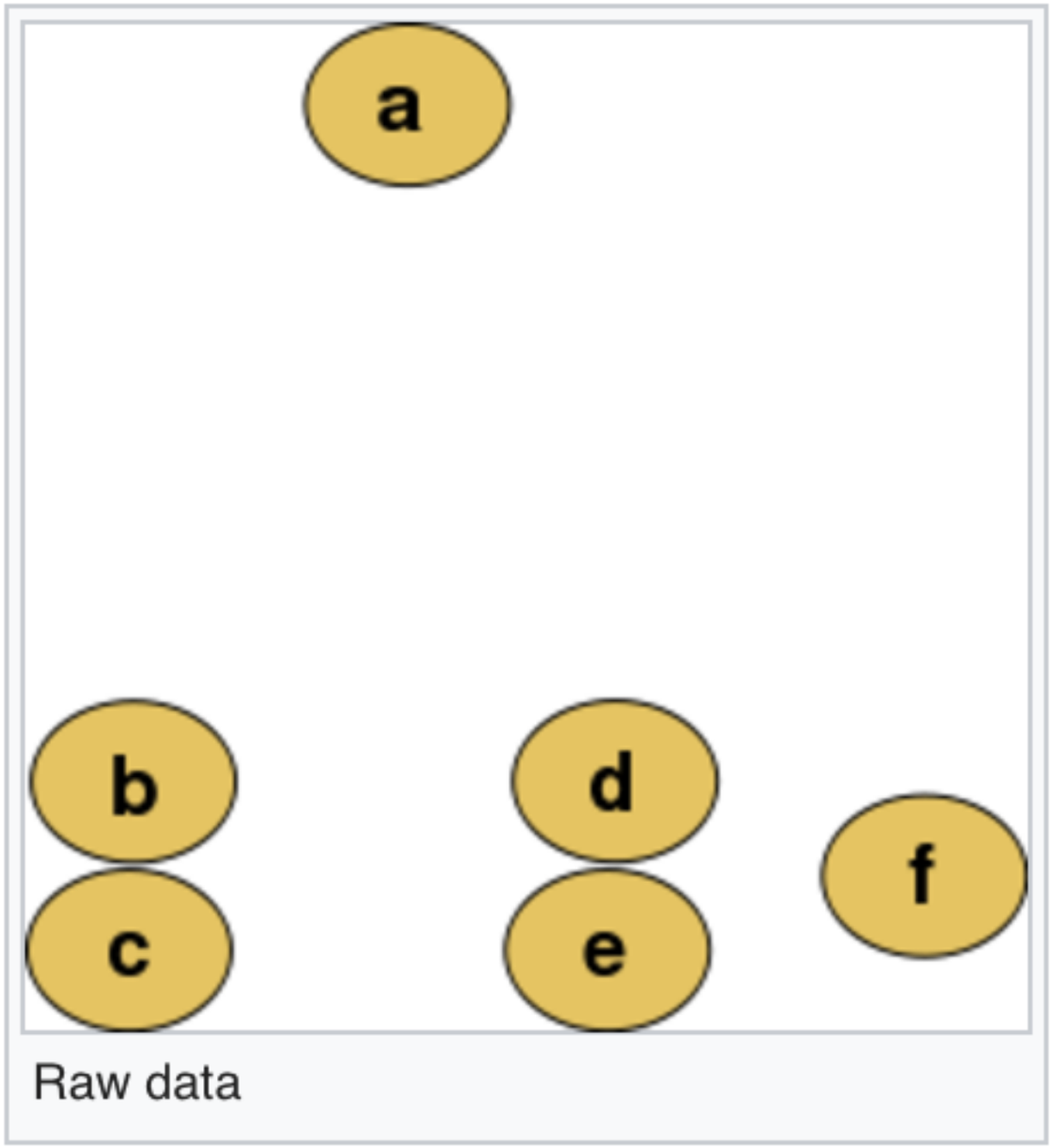
Hierarchical Clustering

- **Agglomerative (bottom-up):**
  - Start with each document being a single cluster.
  - Eventually all documents belong to the same cluster.
- **Divisive (top-down):**
  - Start with all documents belong to the same cluster.
  - Eventually each node forms a cluster on its own.
- Does not require the number of clusters ***k*** in advance
- Needs a termination/readout condition
  - The final mode in both Agglomerative and Divisive is of no use.

Hierarchical Agglomerative Clustering (HAC) Algorithm

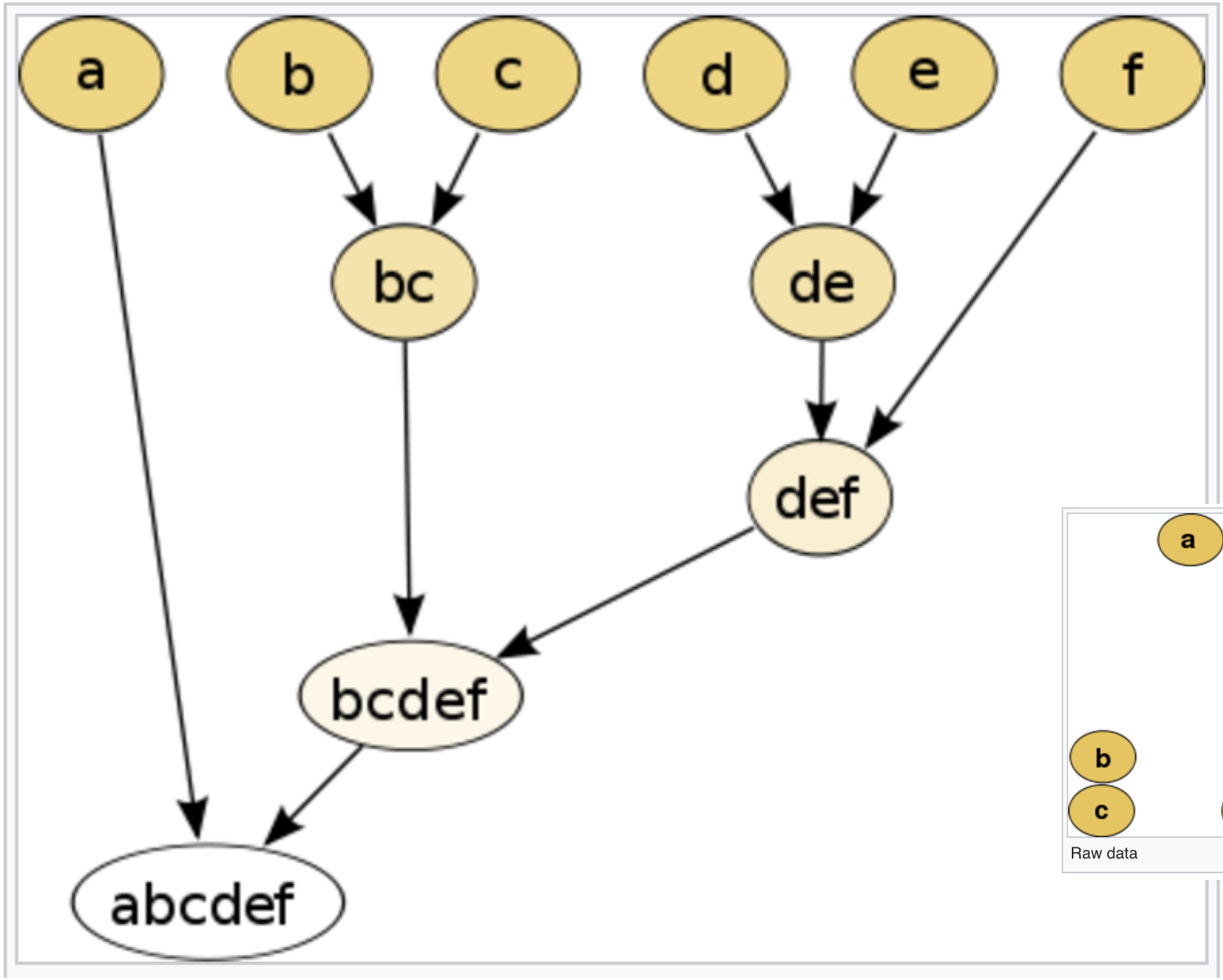
Start with all instances in their own cluster.
Until there is only one cluster:
    Among the current clusters, determine the two clusters, c_i and c_j, that are most similar.
    Replace c_i and c_j with a single cluster $c_i \cup c_j$

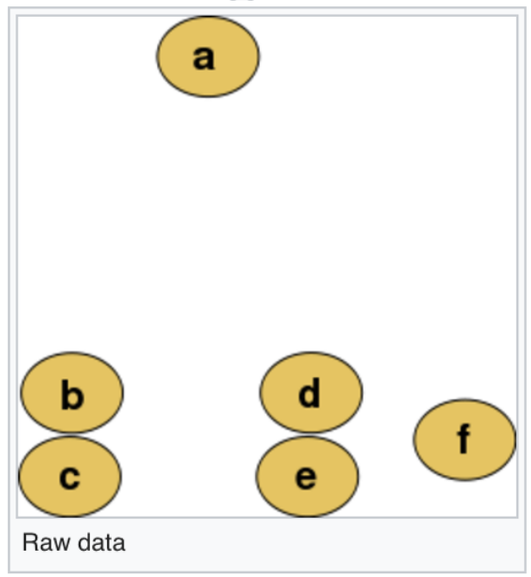
Data points



Raw data

Agglomerative Dendrogram

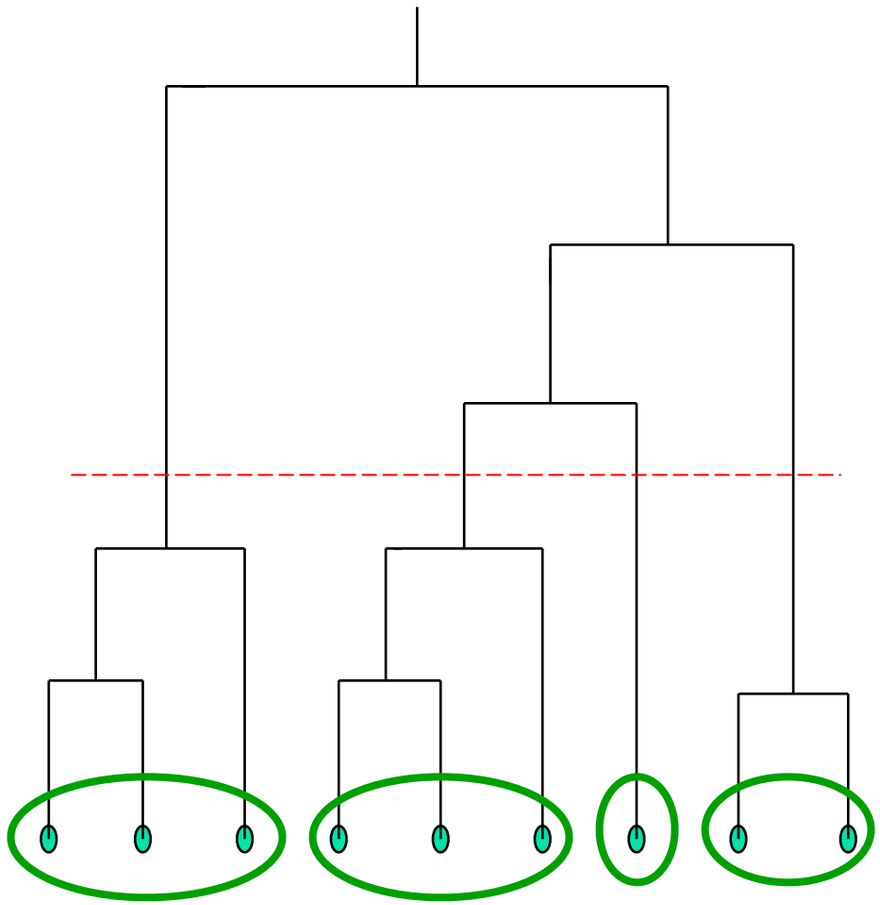




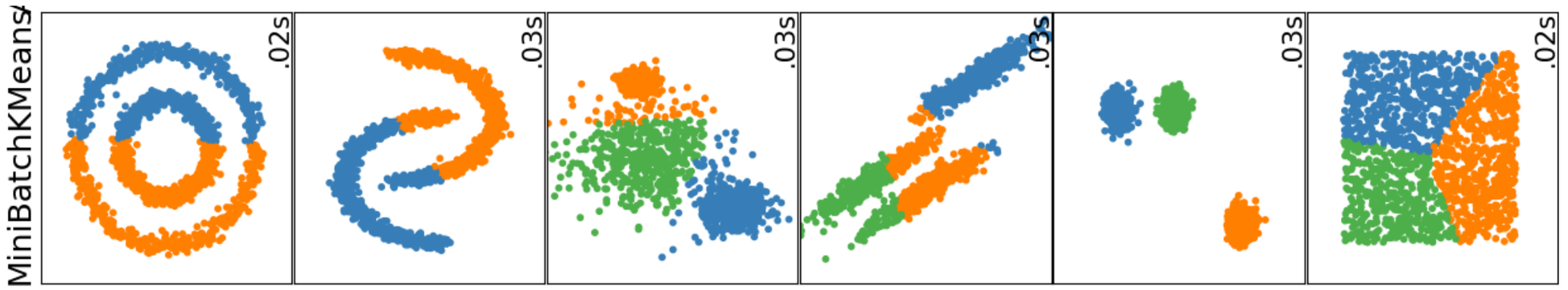
Raw data

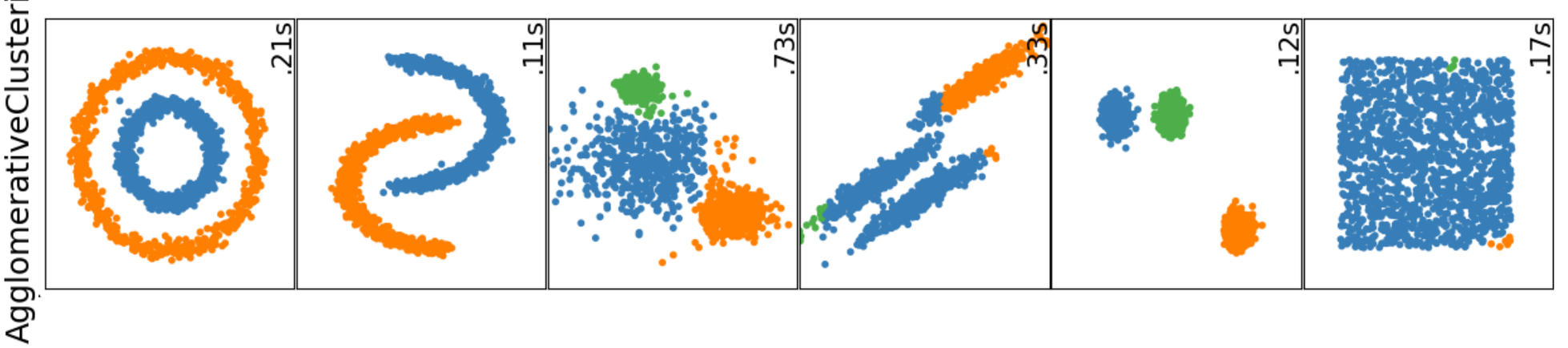
Cutting level of Dendrogram

- Clustering obtained by cutting the dendrogram at a desired level: each connected component forms a cluster.



Agglomerative Clustering





Run Time

- Time complexity : $O(n^3)$
- Memory : $O(n^2)$
- *With heap : $O(n^2 \log n)$*

Distance Metric

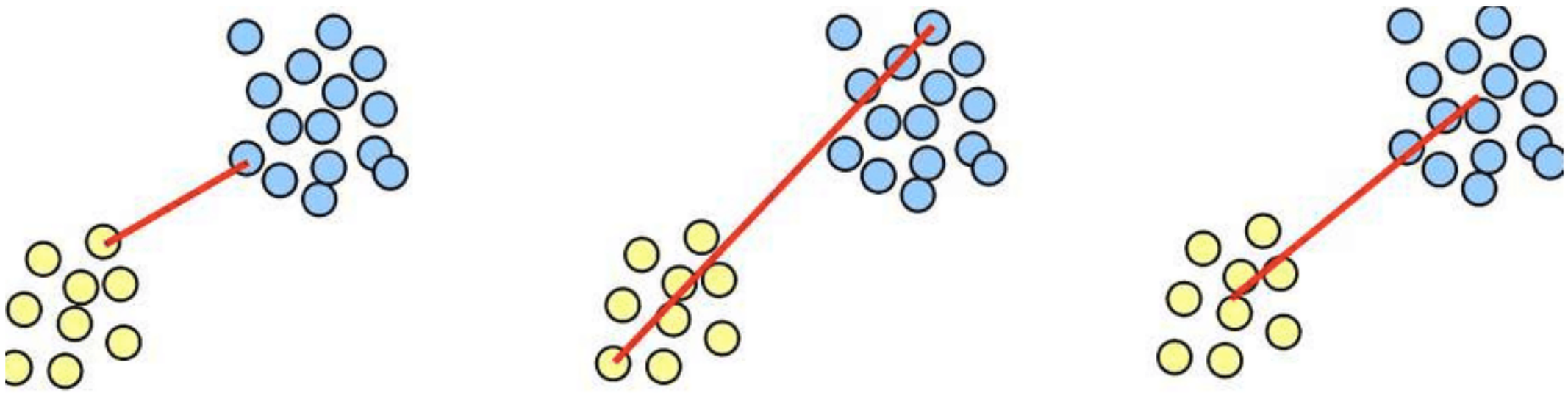
Names	Formula
Euclidean distance	$\|a-b\|_2 = \sqrt{\sum_i (a_i - b_i)^2}$
Squared Euclidean distance	$\|a-b\|_2^2 = \sum_i (a_i - b_i)^2$
Manhattan distance	$\|a-b\|_1 = \sum_i \|a_i - b_i\|$
maximum distance	$\|a-b\|_\infty = \max_i \|a_i - b_i\|$
Mahalanobis distance	$\sqrt{(a-b)^\top S^{-1} (a-b)}$ where S is the Covariance matrix

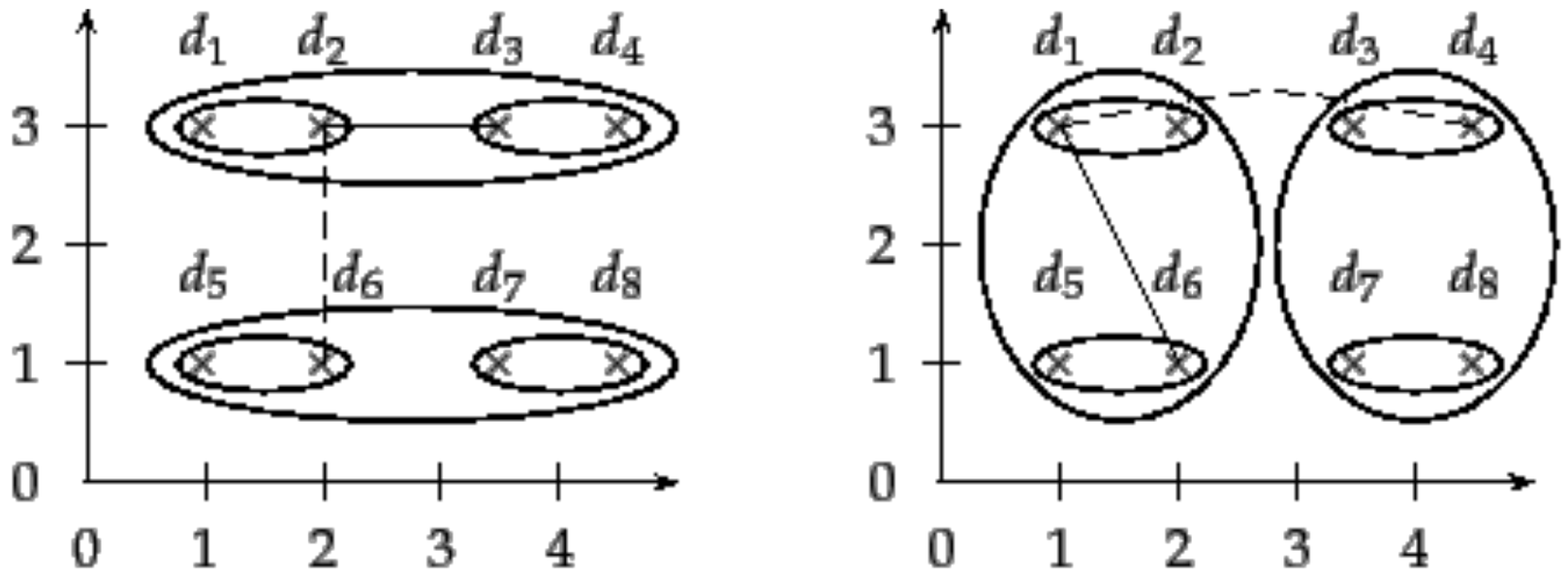
Linkage

Names	Formula
Maximum or complete-linkage clustering	$\max \{\, d(a,b) : a \in A,\, b \in B \,\}.$
Minimum or single-linkage clustering	$\min \{\, d(a,b) : a \in A,\, b \in B \,\}.$
Mean or average linkage clustering, or UPGMA	$\frac{1}{\|A\|\|B\|} \sum_{a \in A} \sum_{b \in B} d(a,b).$
Centroid linkage clustering, or UPGMC	$\|c_s - c_t\|$ where c_s and c_t are the centroids of clusters *s* and *t*, respectively.
Minimum energy clustering	$\frac{2}{nm} \sum_{i,j=1}^{n,m} \|a_i - b_j\|_2 - \frac{1}{n^2} \sum_{i,j=1}^{n} \|a_i - a_j\|_2 - \frac{1}{m^2} \sum_{i,j=1}^{m} \|b_i - b_j\|_2$

Unweighted **P**air **G**roup **M**ethod with **A**rithmetic Mean

Mean is difficult to compute for categorical data

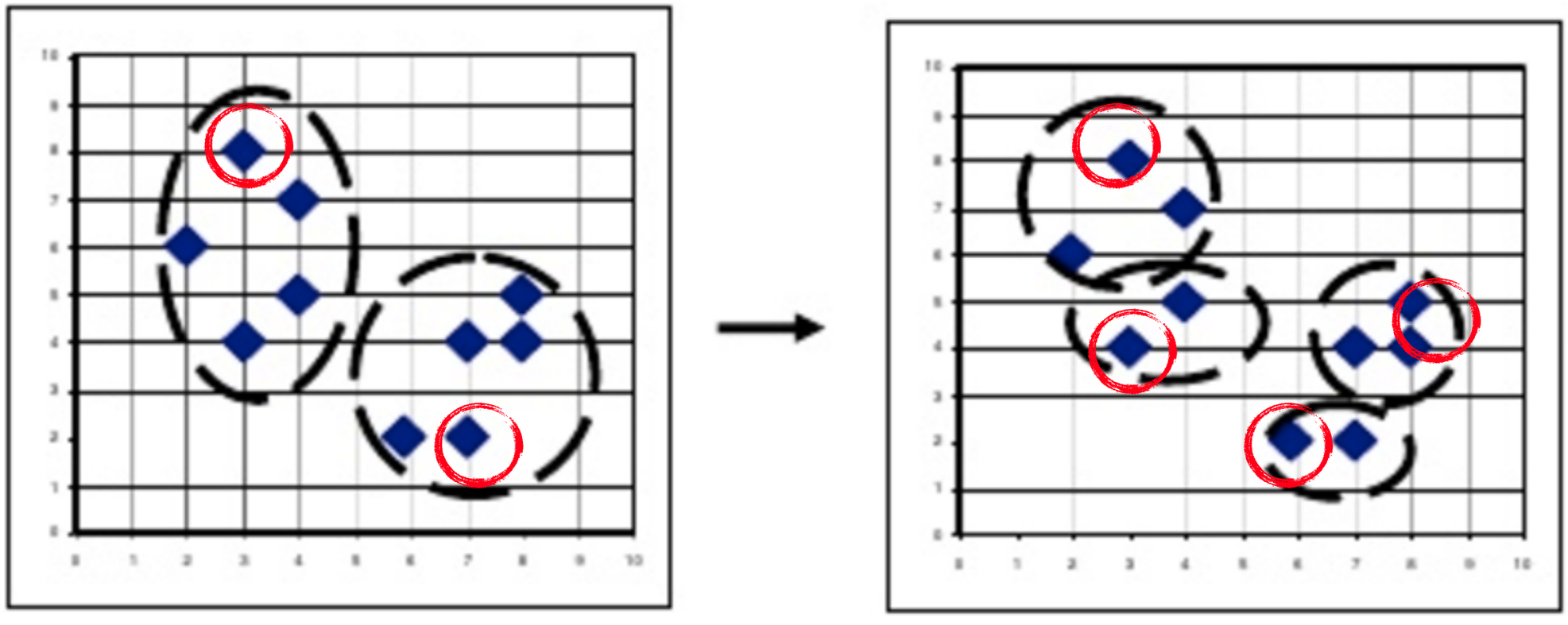




► **Figure 17.2** A single-link (left) and complete-link (right) clustering of eight documents. The ellipses correspond to successive clustering stages. Left: The single-link similarity of the two upper two-point clusters is the similarity of d_2 and d_3 (solid line), which is greater than the single-link similarity of the two left two-point clusters (dashed line). Right: The complete-link similarity of the two upper two-point clusters is the similarity of d_1 and d_4 (dashed line), which is smaller than the complete-link similarity of the two left two-point clusters (solid line).

DIANA (DIvisive ANAlysis Clustering)

Find object which have maximum distance between them
Cluster all object depending upon their closeness to these two selected objects



Issues

- **Lack of a Global Objective Function:** agglomerative hierarchical clustering techniques perform clustering on a local level and as such there is no global objective function like in the K-Means algorithm.

- **No Backtracking:** a particular merge or split turns out to be poor choice, it cannot be corrected.

- **Deciding level of clustering :** subjective decision

- **Complexity** : Time Complexity: $O(n^2 \log n)$

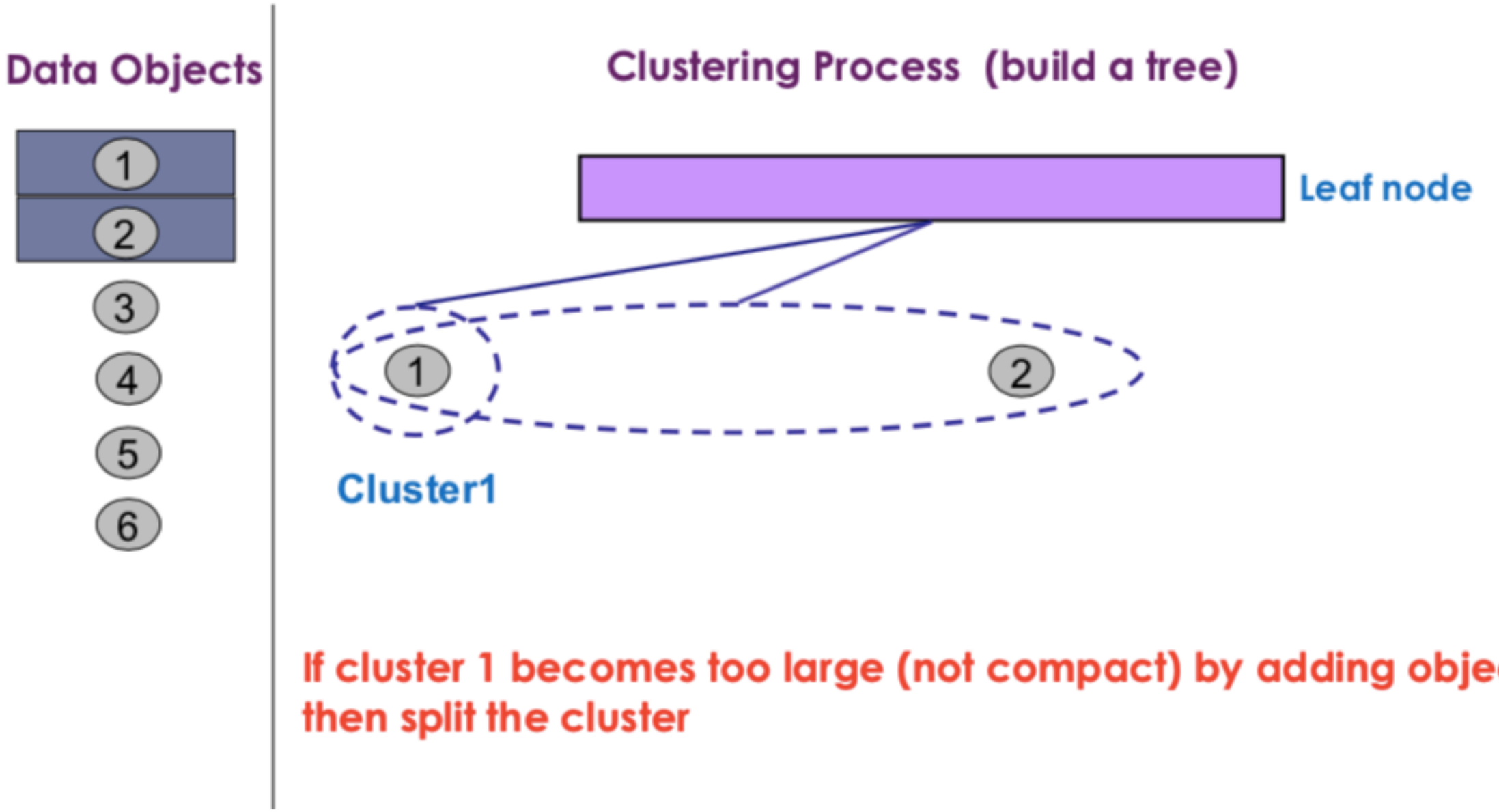
BIRCH

Sahely Bhadra

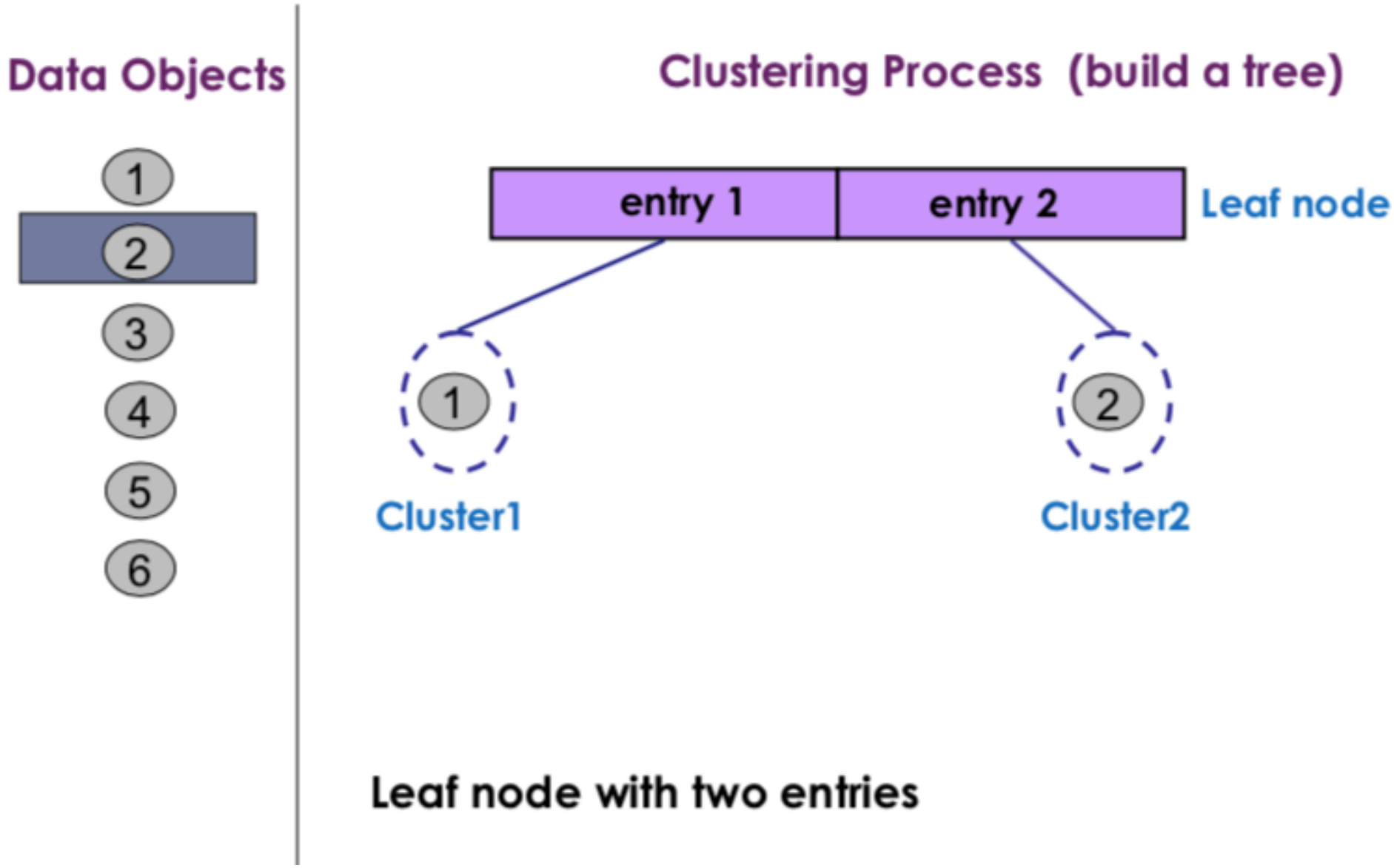
BIRCH

- BIRCH: Balanced Iterative Reducing and Clustering Using Hierarchies
- Agglomerative Clustering designed for clustering a large amount of numerical data
- What Birch algorithm tries to solve?
  - Most of the existing algorithms DO NOT consider the case that datasets can be **too large to fit in main memory**
  - They DO NOT concentrate on **minimizing the number of scans of the dataset** I/O costs are very high
  - The complexity of BIRCH is **O(n)** where n is the number of objects to be clustered.

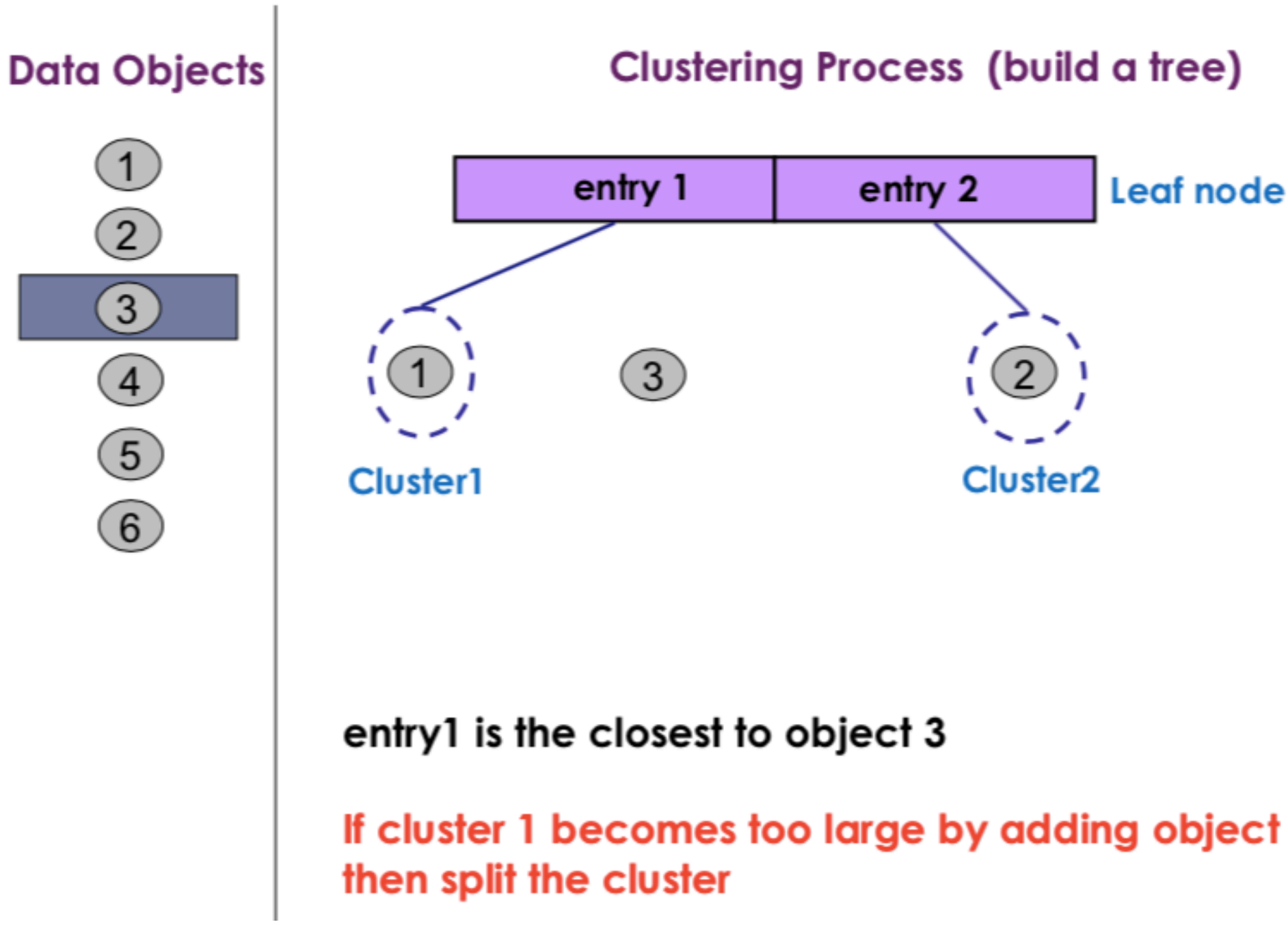
BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
Leaf node
1
2
Cluster1
If cluster 1 becomes too large (not compact) by adding object 2, then split the cluster

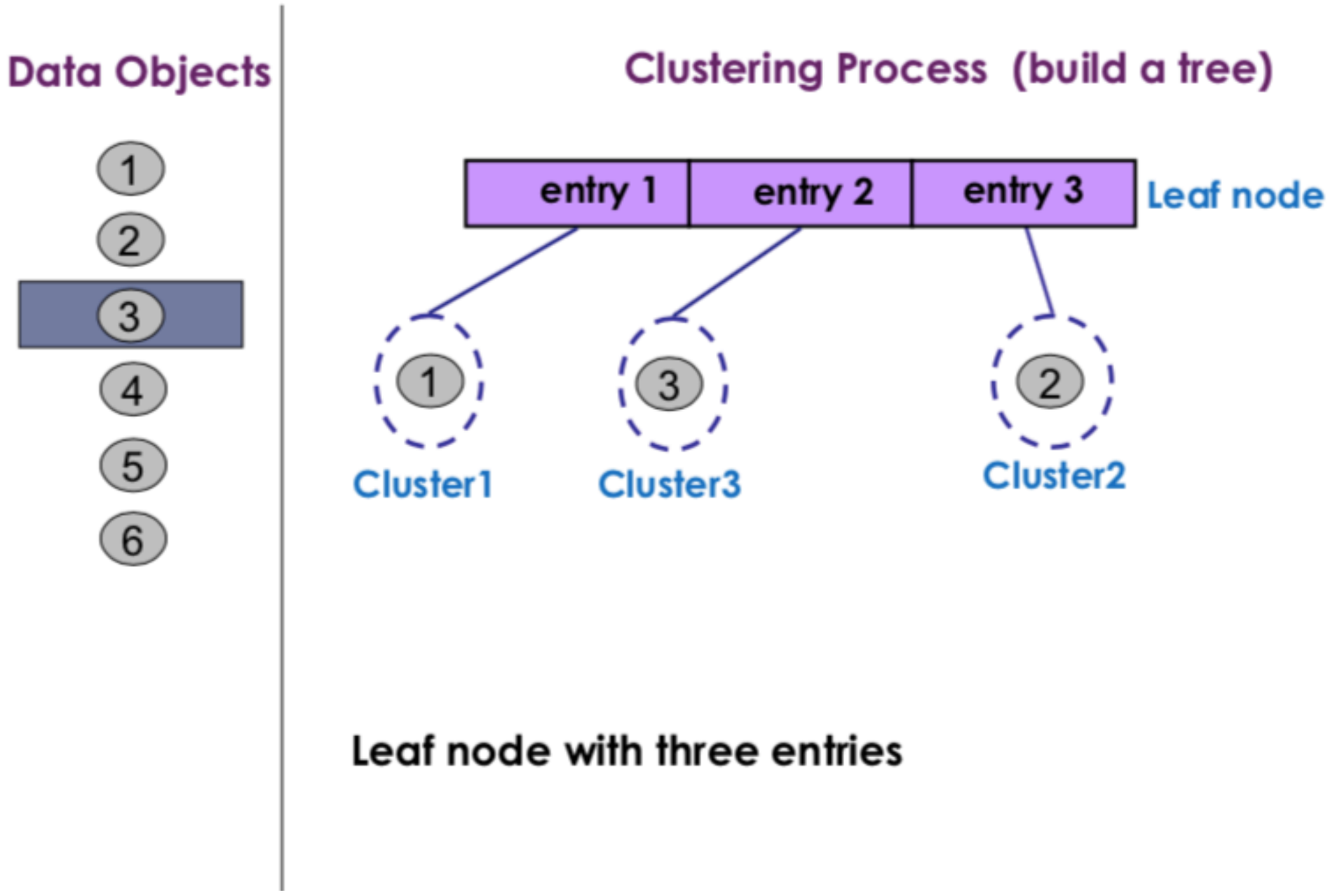
BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
entry 1
entry 2
Leaf node
1
2
Cluster1
Cluster2
Leaf node with two entries

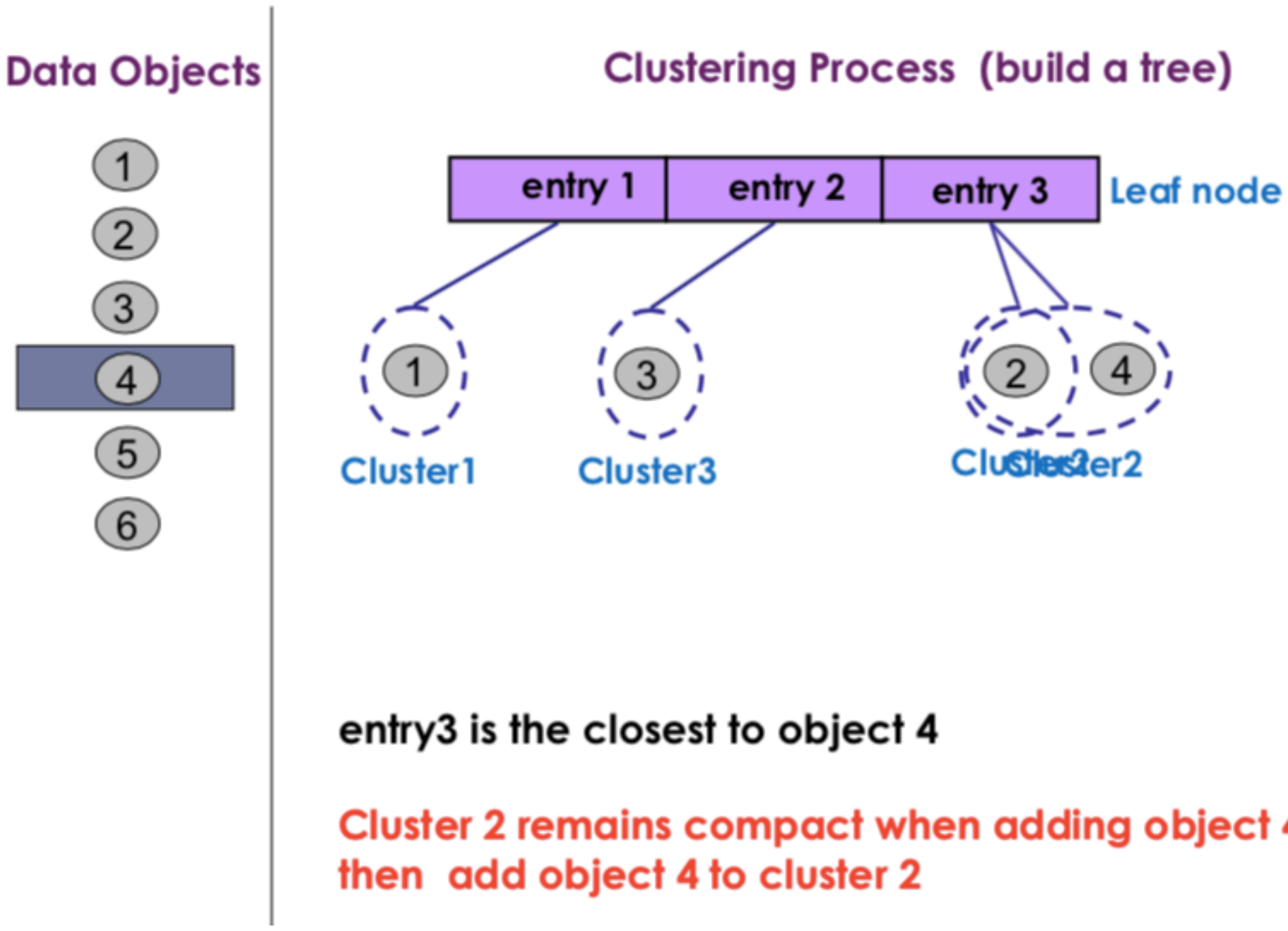
BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
entry 1
entry 2
Leaf node
1
3
2
Cluster1
Cluster2
entry1 is the closest to object 3
If cluster 1 becomes too large by adding object 3, then split the cluster

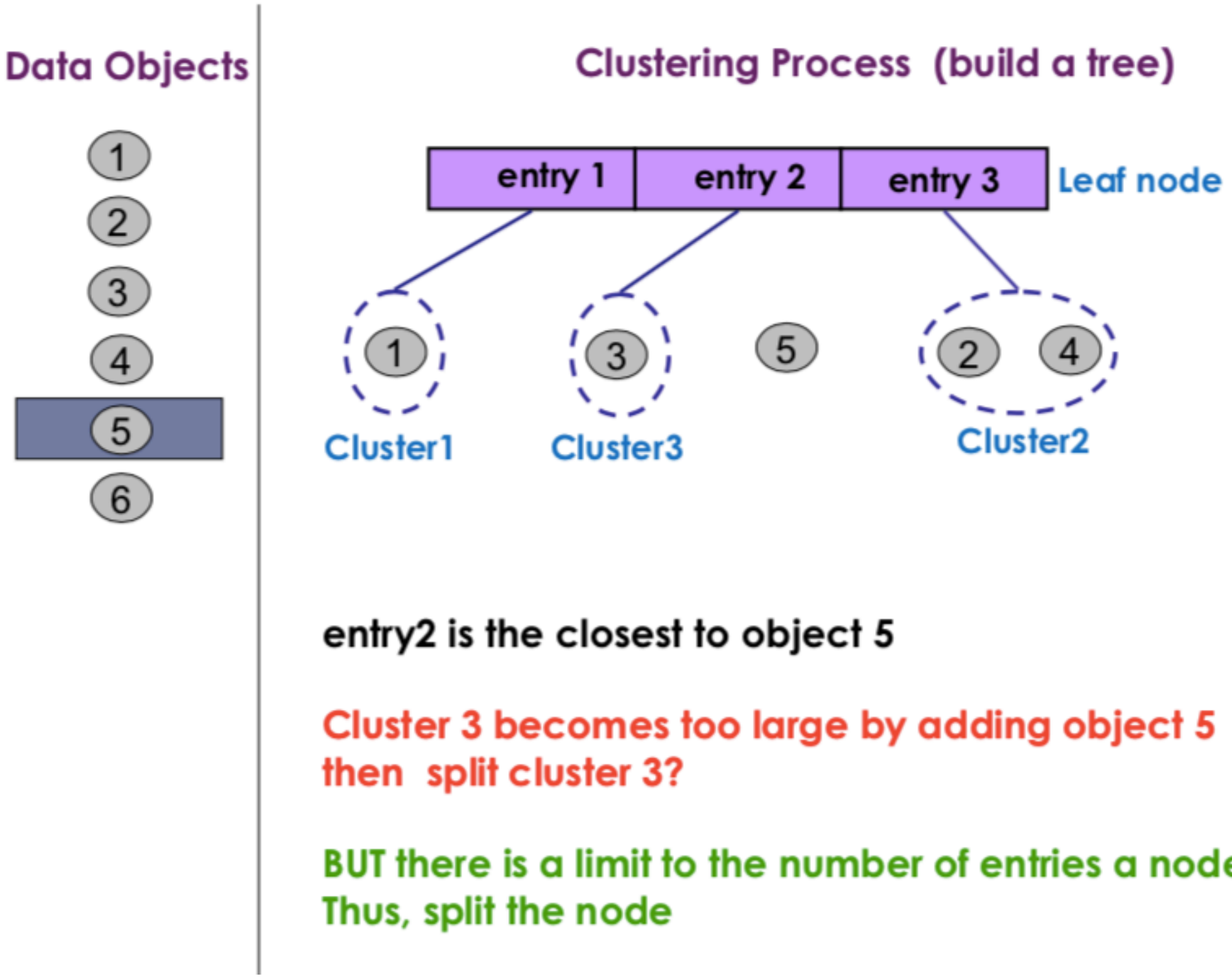
BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
entry 1
entry 2
entry 3
Leaf node
1
3
2
Cluster1
Cluster3
Cluster2
Leaf node with three entries

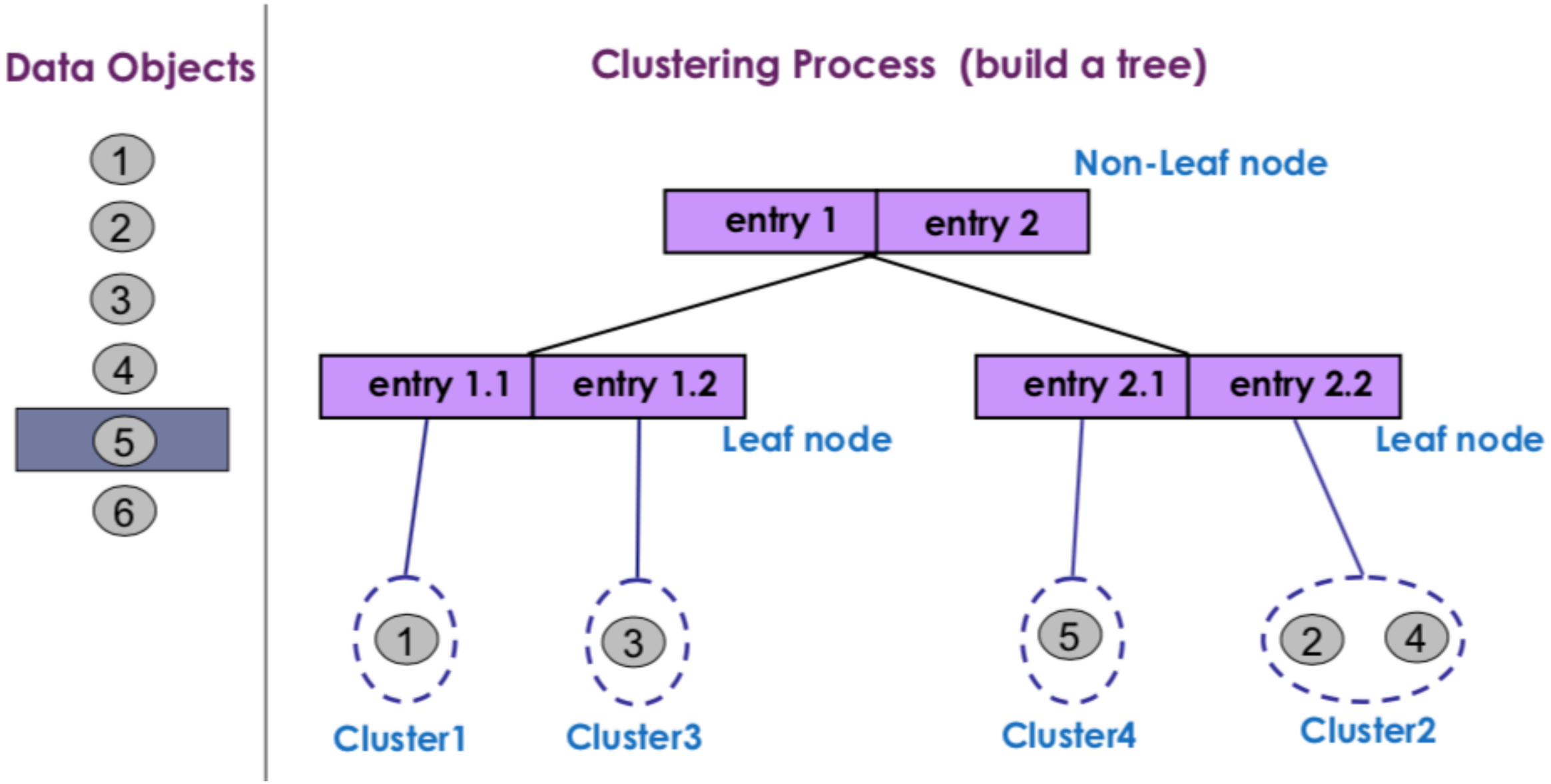
BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
entry 1
entry 2
entry 3
Leaf node
1
3
2
4
Cluster1
Cluster3
Cluster2
entry3 is the closest to object 4
Cluster 2 remains compact when adding object 4
then add object 4 to cluster 2

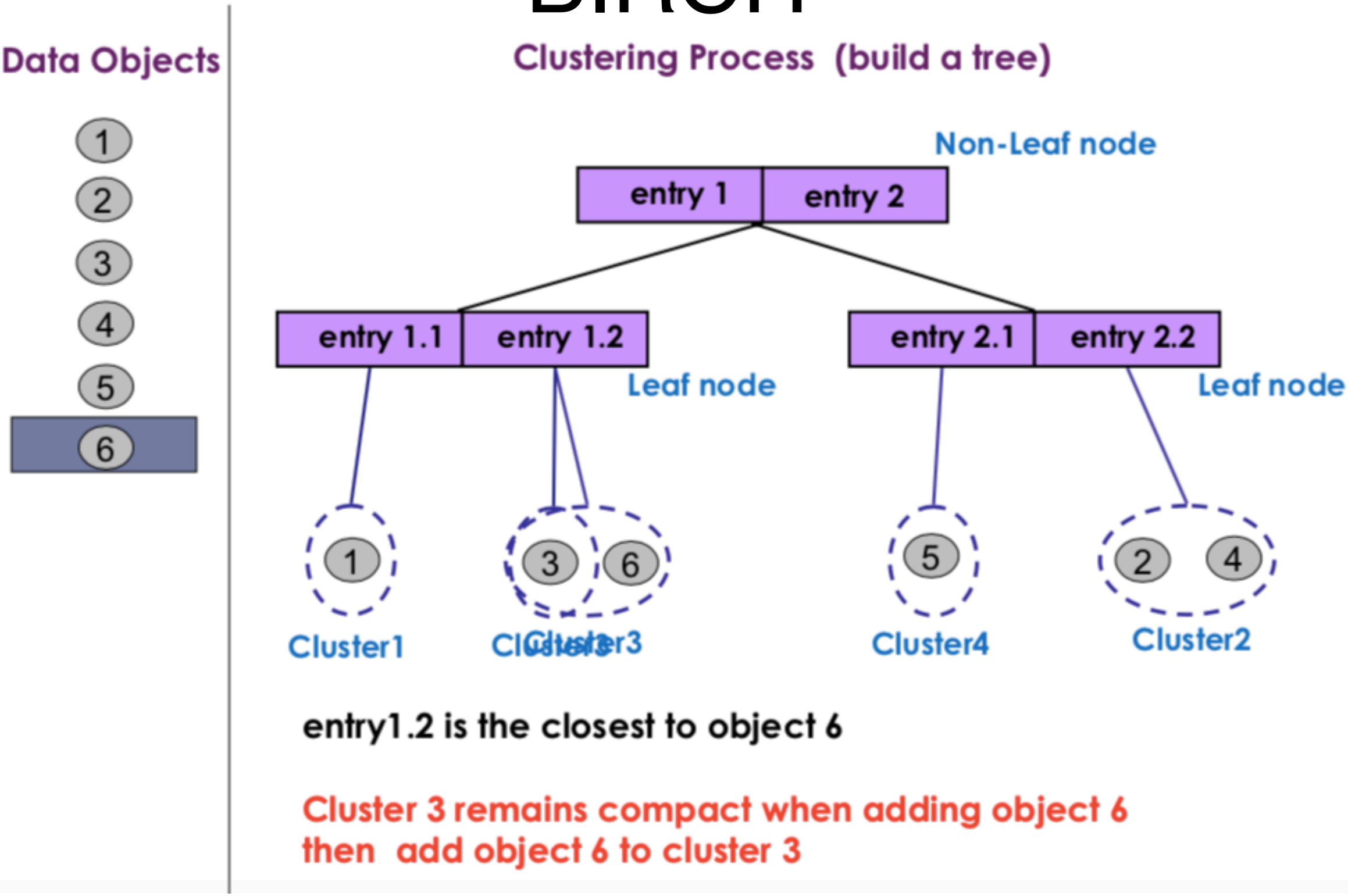
BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
entry 1
entry 2
entry 3
Leaf node
1
3
5
2
4
Cluster1
Cluster3
Cluster2
entry2 is the closest to object 5
Cluster 3 becomes too large by adding object 5
then split cluster 3?
BUT there is a limit to the number of entries a node can have
Thus, split the node

BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
Non-Leaf node
entry 1
entry 2
entry 1.1
entry 1.2
entry 2.1
entry 2.2
Leaf node
Leaf node
1
3
5
2
4
Cluster1
Cluster3
Cluster4
Cluster2

BIRCH


Data Objects
1
2
3
4
5
6
Clustering Process (build a tree)
Non-Leaf node
entry 1
entry 2
entry 1.1
entry 1.2
Leaf node
entry 2.1
entry 2.2
Leaf node
1
3
6
5
2
4
Cluster1
Cluster3
Cluster4
Cluster2
entry1.2 is the closest to object 6
Cluster 3 remains compact when adding object 6
then add object 6 to cluster 3

BIRCH

Clustering Feature (CF)

- Summary of the statistics for a given cluster: the 0-th, 1st and 2nd
- moments of the cluster from the statistical point of view
- Used to compute centroids, and measure the compactness and distance of clusters

CF-Tree

- height-balanced tree
- two parameters:
  - number of entries in each node
  - The *diameter* of all entries in a leaf node
- Leaf nodes are connected via *prev* and *next* pointers

Clustering Features

Clustering features(CF) are organised in a ***CF tree***

- Entries for each child : [$\mathbf{CF_i}$, $\mathbf{Child_i}$]
- **CF = (N, LS, SS)**

N: Number of data points

LS: linear sum of N points $\sum_{i=1}^{N} X_i$

SS: square sum of N points $\sum_{i=1}^{N} X_i^2$

CF3=CF1+CF2= ⟨3+3, (9+35, 10+36), (29+417 , 38+440)⟩ = ⟨6, (44,46), (446 ,478)⟩

CF_1= ⟨3, (2+3+4 , 5+2+3), (2^2+3^2+4^2 , 5^2+2^2+3^2)⟩ = ⟨3, (9,10), (29 ,38)⟩

Clustering Features

Clustering features(CF) are organised in a ***CF tree***

- Entries for each child : [**CF_i, $Child_i$**]
- **CF = (N, LS** $\sum_{i=1}^{N} X_i$

N: Number of data points

LS*:* linear sum of N points= $\sum_{i=1}^{N} X_i^2$

SS*:* square sum of N points=

CF entry is a **summary** of statistics of the cluster

A **representation** of the cluster

A CF entry has **sufficient information** to calculate the centroid, radius, diameter and many other distance measures

Additively theorem allows us to **merge sub-clusters incrementally**

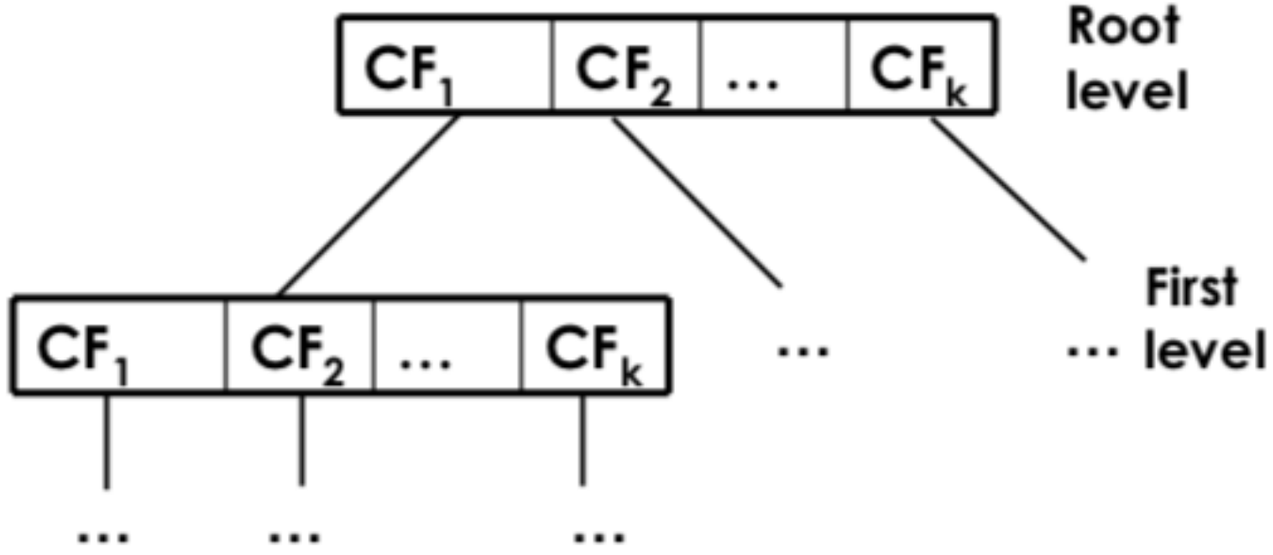
.

CF Tree

B = Branching Factor, maximum children in a non-leaf node

T = Threshold for diameter or radius of the cluster in a leaf

L = number of entries in a leaf



- CF entry in parent = sum of CF entries of a child of that entry
- In-memory, height-balanced tree
-

CF Tree Insertion

- Start with the root
- Find the CF entry in the root closest to the data point, move to that child and repeat the process until a closest leaf entry is found.
- At the leaf
    - If the point can be accommodated in the cluster, update the entry
    - If this addition violates the threshold T, split the entry,
        - if this violates the limit imposed by L, split the leaf.
            - If its parent node is full, split that and so on
- Update the CF entries from the leaf to the root to accommodate this point
-

DBSCAN

Sahely Bhadra

Density Based Clustering

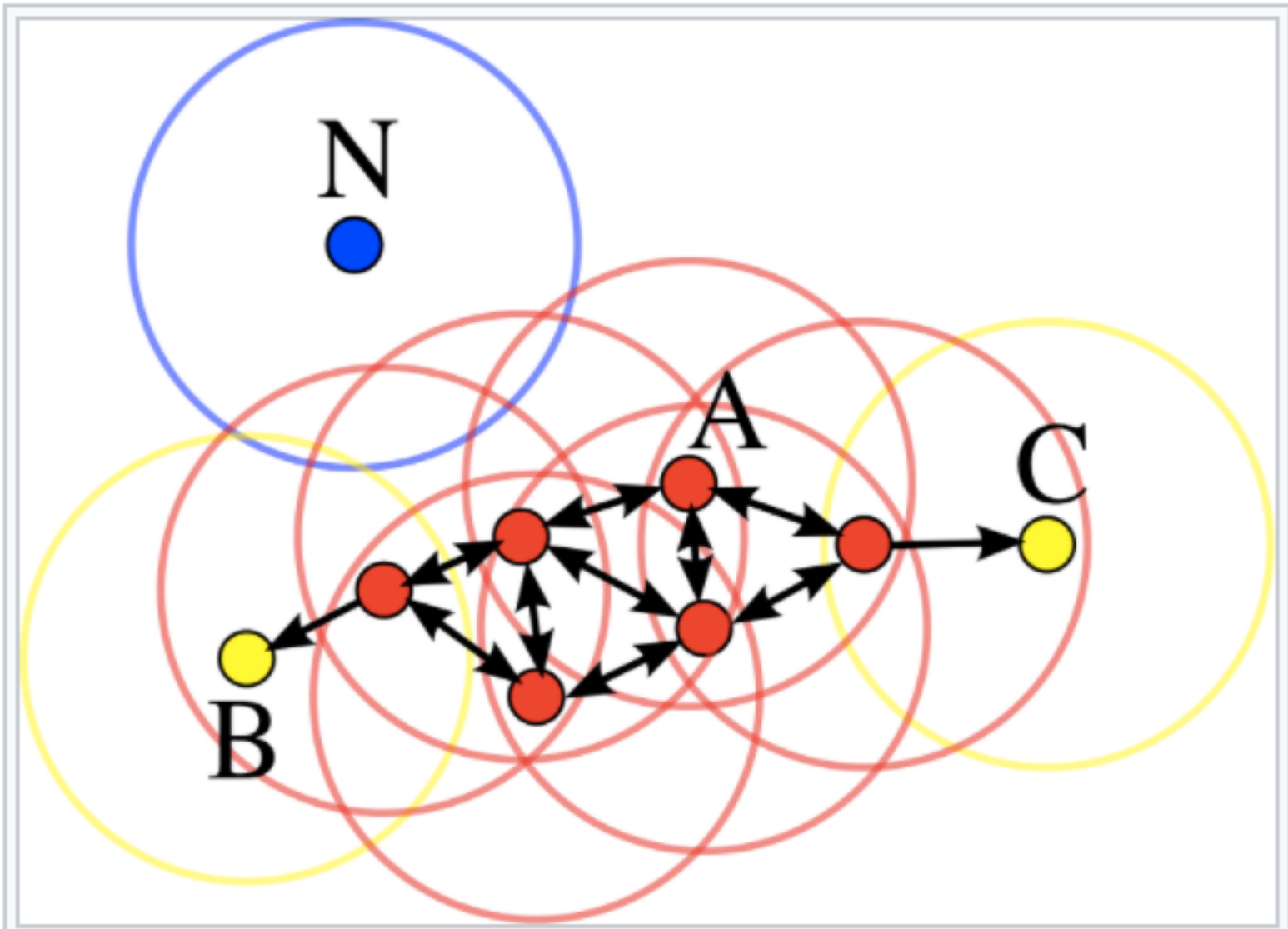
- Clustering based on density (local cluster criterion), such as density-connected points or based on an explicitly constructed density function
- Major features:
  - Discover clusters of arbitrary shape
  - Handle noise
  - One scan
  - Need density parameters

- Several interesting studies:
  - DBSCAN: Ester, et al. (KDD'96)
    Density-based spatial clustering of applications with noise
  - DENCLUE: Hinneburg & D. Keim (KDD'98/2006)
  - OPTICS: Ankerst, et al (SIGMOD'99).
  - CLIQUE: Agrawal, et al. (SIGMOD'98)

DBSCAN: Density-Based Algorithm for Discovering Clusters

- DBSCAN is a density-based algorithm.
  - Density = number of points within a specified radius r (Eps)
  - A point is a core point if it has more than a specified number of points (MinPts) within Eps
    - These are points that are at the interior of a cluster
  - A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point
  - A noise point is any point that is not a core point or a border point.

DBSCAN: Density-Based Algorithm for Discovering Clusters



In this diagram, $\mathrm{minPts} = 4$. Point A and the other red points are core points, because the area surrounding these points in an ε radius contain at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor directly-reachable.

```
DBSCAN(DB, distFunc, eps, minPts) {
   C = 0                                                  /* Cluster counter */
   for each point P in database DB {
      if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
      Neighbors N = RangeQuery(DB, distFunc, P, eps)      /* Find neighbors */
      if |N| < minPts then {                              /* Density check */
         label(P) = Noise                                 /* Label as Noise */
         continue
      }
      C = C + 1                                           /* next cluster label */
      label(P) = C                                        /* Label initial point */
      Seed set S = N \ {P}                                /* Neighbors to expand */
      for each point Q in S {                             /* Process every seed point */
         if label(Q) = Noise then label(Q) = C            /* Change Noise to border point */
         if label(Q) ≠ undefined then continue            /* Previously processed */
         label(Q) = C                                     /* Label neighbor */
         Neighbors N = RangeQuery(DB, distFunc, Q, eps)   /* Find neighbors */
         if |N| ≥ minPts then {                           /* Density check */
            S = S ∪ N                                     /* Add new neighbors to seed set */
         }
      }
   }
}
```

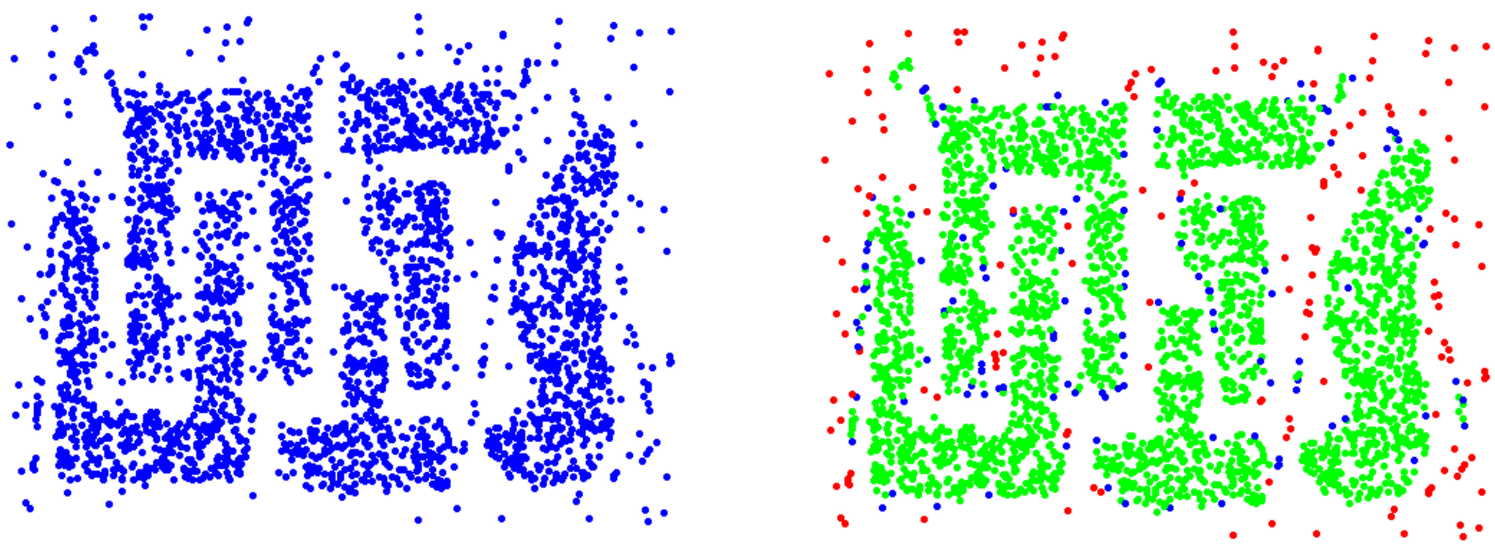
```
RangeQuery(DB, distFunc, Q, eps) {
   Neighbors = empty list
   for each point P in database DB {                      /* Scan all points in the database */
      if distFunc(Q, P) ≤ eps then {                      /* Compute distance and check epsilon */
         Neighbors = Neighbors ∪ {P}                      /* Add to result */
      }
   }
   return Neighbors
}
```

Complexity

- Time Complexity: $O(n^2)$—for each point it has to be determined if it is a core point, can be reduced to O(n*log(n)) in lower dimensional spaces by using efficient data structures (n is the number of objects to be clustered);

- Space Complexity: O(n).

DBSCAN: Density-Based Algorithm for Discovering Clusters

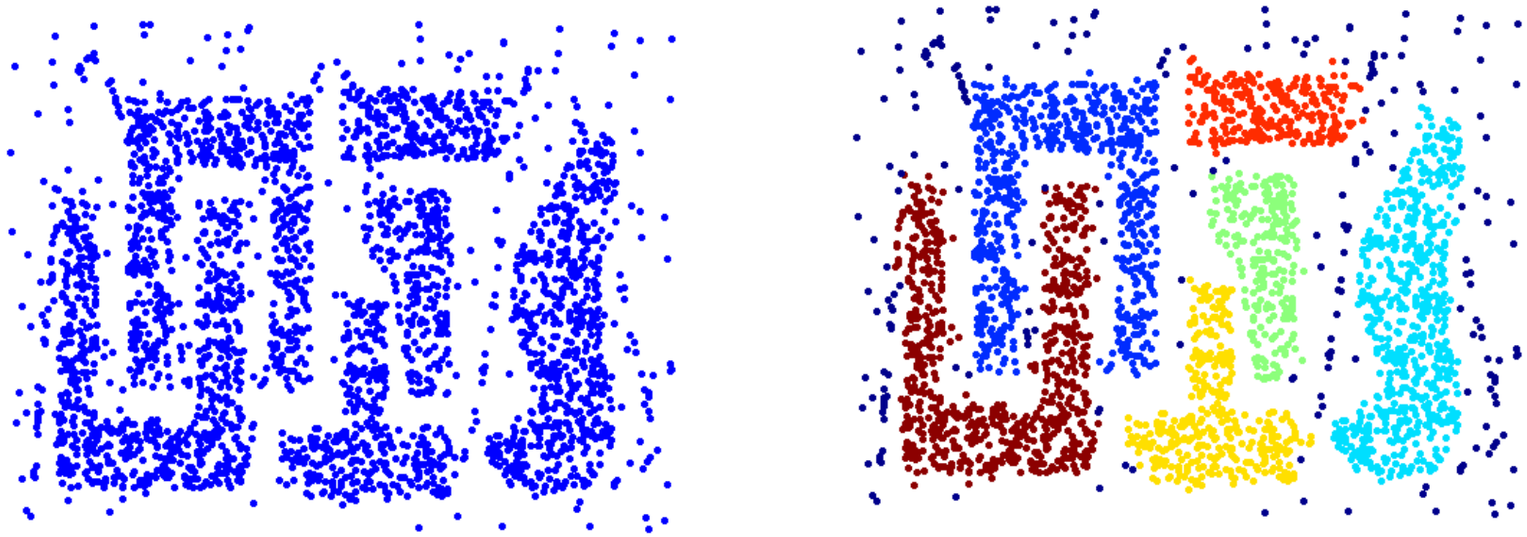


Original Points

Point types: core, border and noise

Eps = 10, MinPts = 4

When DBSCAN works well?

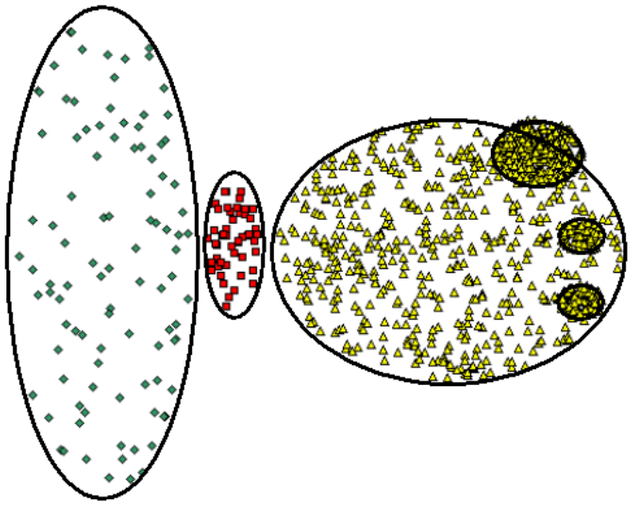


Original Points

Clusters

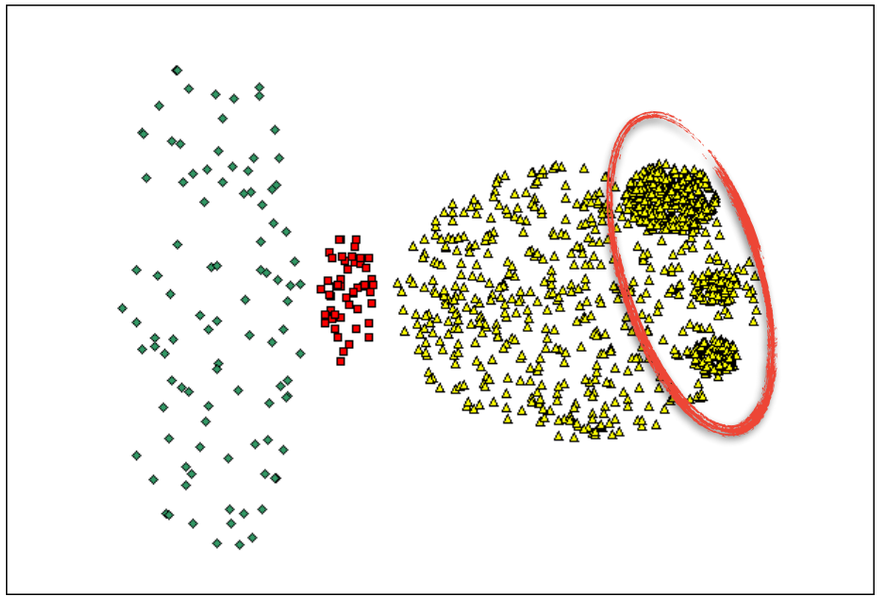
- Resistant to Noise
- Can handle clusters of different shapes and sizes

When DBSCAN fails?



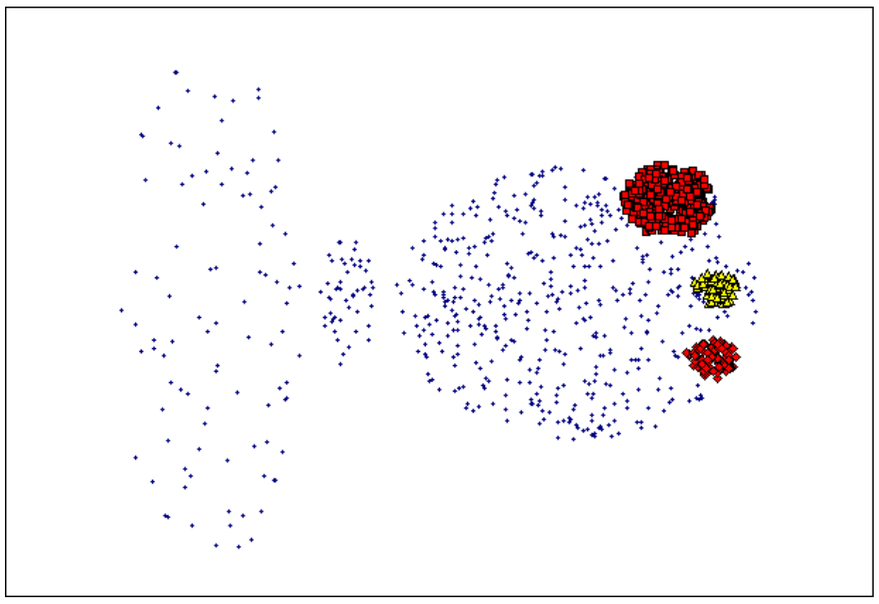
Original Points

- Varying densities
- High-dimensional data



(MinPts=4, Eps=9.75).

Miss these clusters



(MinPts=4, Eps=9.92)

Summery

- Good:
  - can detect arbitrary shapes,
  - not very sensitive to noise,
  - supports outlier detection,
  - complexity is kind of okay,
  - beside K-means the second most used clustering algorithm.

- Bad:
  - does not work well in high-dimensional datasets,
  - parameter selection is tricky,
  - has problems of identifying clusters of varying densities